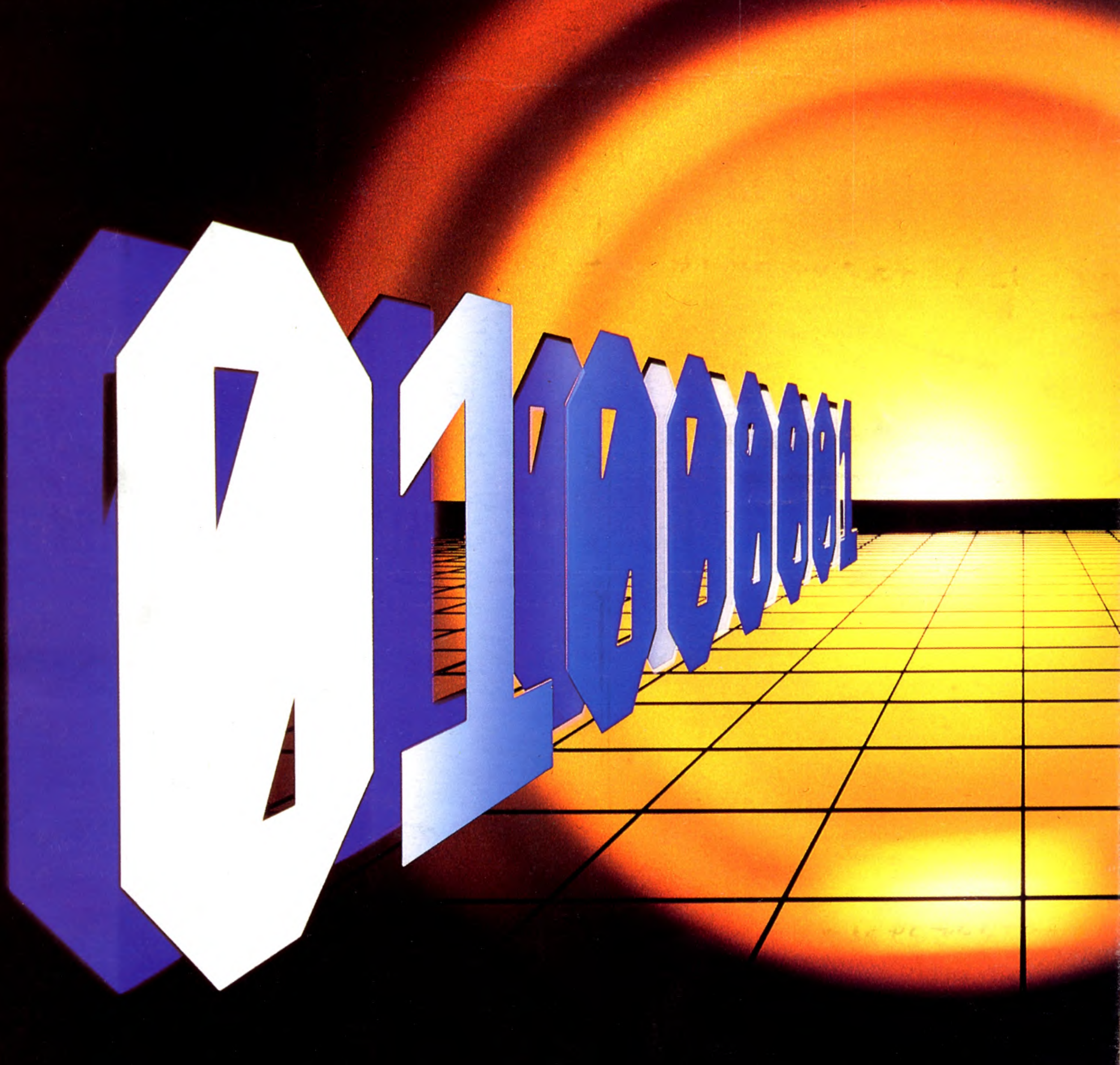
A MARSHALL CAVENDISH 30 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

INPUT

Vol. 3 No 30

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Paul Chave, Ian Stephen. Pages 921, 922, 923, 926, Kuo Kang Chen. Pages 928, 931, Paul Chave, Ian Stephen. Pages 933, 934, 935, 936, 937, Paddy Mounter. Pages 940, 941, 942, 943, 945, Kevin O'Keefe. Page 946, Dave King. Pages 948, 950, Dave King, Chris Lyon.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN
Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015
New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington
Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).
UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN
Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to: *Marshall Cavendish Partworks Ltd.*

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and + — COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+ — DRAGON 32 and 64
ZX81 — VIC 20 — TANDY TRS80 COLOUR COMPUTER

SUPERCHARGE YOUR BASIC

- MAKING BASIC PROGRAMS RUN FASTER
- TIMING BASIC STATEMENTS
- LOOKING AT STRUCTURE, MEMORY USE AND SPEED

Don your crash hats, firesuits and visors. Clean your plugs, and make sure your machine is running sweetly. The flag is about to drop, for it's time to speed up BASIC

At first, all computers seem lightning fast, performing tasks 'in a blink of an eye' which would take a human very much longer. But try to write an action game, or get the machine to perform a lengthy series of calculations, or a complex sorting task, using BASIC and you'll find it's another story altogether. The machine certainly *does* take time to complete the allotted task. Soon, you'll find yourself complaining about how *slow* your machine is!

The fastest running programs are those written in machine code or assembly language—see *INPUT*'s Machine Code programming strand—but most people find it far easier to program in BASIC. Unfortunately, a program written in BASIC can never hope to approach the speed of a program written in machine code because the computer has to spend time translating the program from BASIC into machine code. It has a special program installed as part of its hardware which does this job, called an *interpreter*.

If you do not wish to write programs in machine code, but still want to extract the last ounce of speed from the machine, there are various ploys that can be tried. First, you should try to structure your programs properly—see pages 173 to 178. Second, you should try to make each individual program line operate at the optimum speed. Choose to use the parts of BASIC which the interpreter is quickest at translating.

Every machine has its own quirks, and to some extent, each program its own requirements. Consequently, there can be no hard-and-fast rules for the perfect program. It is only possible to give general pointers—it is up to you how many of the tips you choose to incorporate into your programs, because there are sacrifices to be made in using them.

TIMING BASIC PROGRAMS

All the machines have a built-in timer which can be used to compare how fast programs



run. Type in the routine below which has been designed for your machine, so you can follow through the examples later in the article, and you can see for yourself the differences between alternative forms of BASIC programming:

```
1 POKE 23672,0: POKE 23673,0: POKE
   23674,0
100 REM SPECTRUM FUNCTIONS TIMER
120 FOR i=1 TO 100
130 GOSUB 200: NEXT i
140 LET b=PEEK 23672+256*PEEK
   23673+65536*PEEK 23674
150 PRINT AT 5,5;(b-41)/5;
   " MILLISECS"
160 STOP
200 REM
500 RETURN
```

```
100 T=TI
110 FORI=1TO100:GOSUB200:NEXTI
120 T=TI-T
130 PRINT"♥TIME TAKEN=";
   (T-21)/6;"□MILLISECS."
140 END
200 REM
500 RETURN
```

```
100 T=TI
110FORI=1TO100:GOSUB200:NEXTI
120 T=TI-T
130 PRINT"♥TIME TAKEN=";
   (T-18)/6;"□MILLISECS."
140 END
200 REM
500 RETURN
```

```
10 REM TIMER FOR BBC
100 TIME=0
101 TIME=TIME-194
110 FOR K=1 TO 2000:GOSUB 200:NEXT
130 CLS:PRINT"□TIME TAKEN□=";
   TIME/100;"□SECONDS"
140 END
200 REM
1000 RETURN
```

```
10 REM TIMER FOR ELECTRON
100 TIME=0
101 TIME=TIME-266
110 FOR K=1 TO 2000:GOSUB 200:NEXT
130 CLS:PRINT"□TIME TAKEN□=";
   TIME/100;"□SECONDS"
140 END
200 REM
1000 RETURN
```

```
100 TIMER=0
110 FORK=1TO600:GOSUB200:NEXT
120 T=TIMER-101
130 CLS:PRINT"□TIME TAKEN=";
   T/50;"SECONDS"
140 END
200 REM
1000 RETURN
```

Each of these programs contains a REM line at Line 200 which will later be used to slot in the test piece. The Spectrum program also contains a REM at Line 100 which is part of the timing correction, and should not be omitted. The REMs in Line 10 of the Acorn programs simply identify which is the BBC version and which the Electron. For the Tandy, the 101 in Line 120 should be replaced by 170.

STRUCTURE

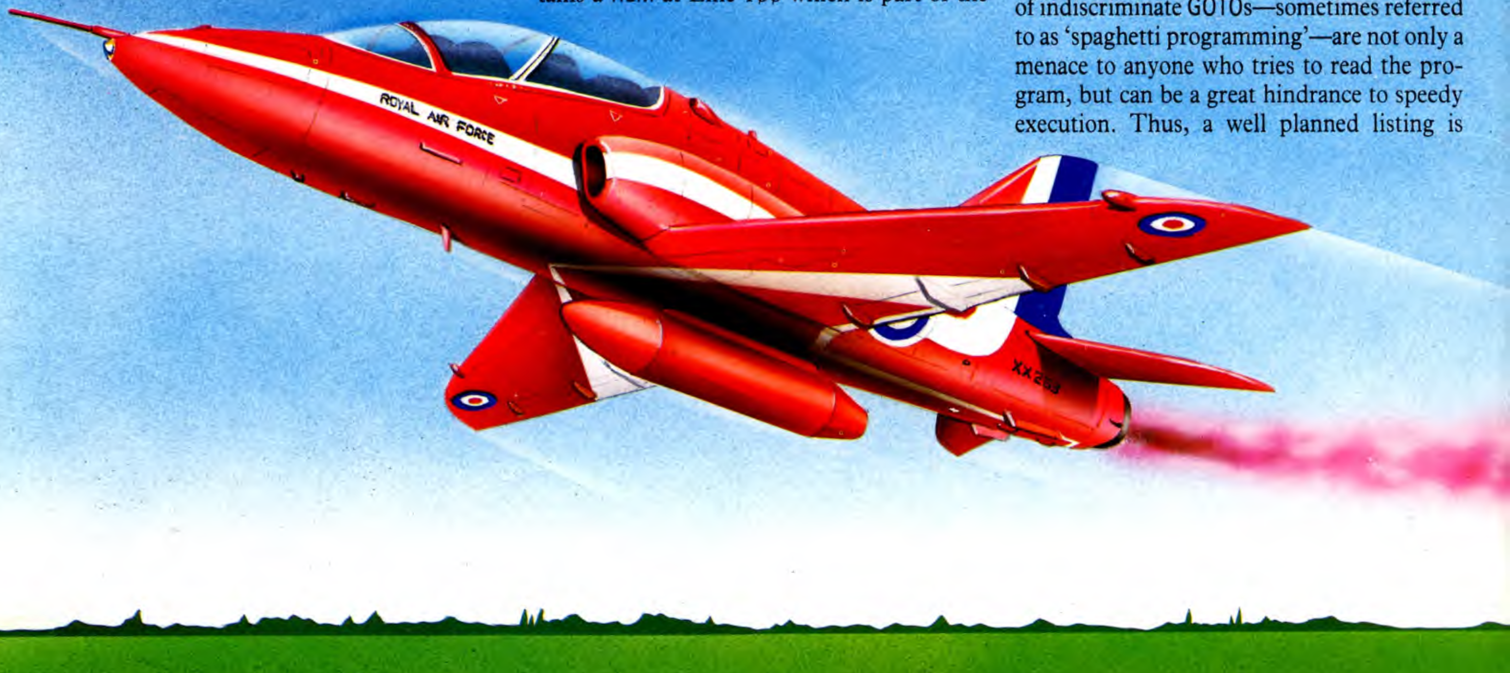
Although a considerable amount has already been said in *INPUT* about structuring programs, it's worth recapping on those factors which have a bearing on the speed of the program.

Place all frequently used subroutines at or near the beginning of the program. This is because the interpreter starts scanning for the GOSUB line number from the beginning. Obviously, if the line number is a low one, the line will be found more quickly than a line with a higher number. A few milliseconds saved here and there can soon make a significant contribution to speed increase.

In addition to subroutines, the Acorn machines have the PROCedures offered by BBC BASIC. Apart from the advantages of structured programming, opting for PROCedures can allow you to write faster-running programs. Although using a PROCedure will take a little longer than a subroutine doing the same thing, the matter isn't quite as simple as all that.

If you have a number of subroutines, they all cannot be sited at the fastest position, right at the start of the program. The further from the start, the slower running the subroutine. On the other hand, PROCedures can be put anywhere you wish in the program, and the speed will not be affected.

Badly thought-out programs using a maze of indiscriminate GOTOs—sometimes referred to as 'spaghetti programming'—are not only a menace to anyone who tries to read the program, but can be a great hindrance to speedy execution. Thus, a well planned listing is



likely to be faster in execution than a program that has just 'evolved'.

MEMORY AND SPEED

Very broadly, shorter programs are also quite likely to run faster. However, the three programmer's aims—speed, clarity and memory economy—are usually in conflict. Multistatement lines, for example, save memory, and speed the program, but can make the listing less easy to follow and debug.

A program designed for maximum speed of running can increase greatly in length and memory usage. Conversely, some otherwise very valid memory-saving techniques save memory at the expense of speed.

A great deal of memory can be saved by using subroutines, but calling subroutines is time-consuming and therefore is undesirable if you are aiming for the very fastest execution. As you have seen already, if you do opt for subroutines, you have to be very careful with the structure of the program to ensure that some of the speed loss is recovered.

Similarly, LET A=VAL "1ØØ" is great for saving memory, but is disastrous when compared with the more memory-hungry LET A=1ØØ when you're aiming for speed.

VARIABLES

A knowledge of how variables are stored in memory can be useful when seeking to speed up programs. The variables area is emptied by RUN or CLEAR, and variables are created as they occur. Generally, new variables are added to extend the variables area upwards—although this doesn't apply to the Sinclair Spectrum.

Consider a situation where you create a string variable, then a numeric array. Later you might add something to the string. If the string has been created early in the program, then all the subsequently created variables will have to be moved up in memory to accommodate any additions to the string.

Take a look at this example:

```
1ØØ LET T$=""
11Ø DIM A(1ØØØ)
12Ø LET T$=T$+"WHATEVER"
```

In most versions of BASIC, you would save a considerable amount of time by exchanging the first two lines of the program, DIMensioning the array before defining the string. In the program as it appears, 5000 bytes might be moved each time "WHATEVER" is added to T$. Spectrum owners need not be concerned, as their BASIC has a different method of managing the variables area, so this problem does not arise.

There is a general rule that if you use variables instead of numbers, then you save a considerable amount of time. A concrete example might come from the Commodore 64. Each variable saves approximately five to ten milliseconds. This may not sound a lot, but if there are a number of loops where numbers are constantly handled, then the time saved could be considerable. In the case of the Spectrum, the opposite seems true. LET C=1Ø+1Ø takes 3 milliseconds, whereas LET C=D+D (where D=1Ø) takes 4.2 milliseconds.

MATHEMATICAL FUNCTIONS

Use the timing routine to compare these alternative forms:

```
2ØØ LET C=4*4*4*4
```

or

```
2ØØ LET C=4↑4
```

You may well expect the machine's own powers function to be the faster of the two. On the Spectrum, for example, the first takes 6 milliseconds, whereas the second takes 114 milliseconds! This is yet another example of memory economy conflicting with speed.

It has sometimes been suggested that on the Spectrum lines like:

```
21Ø IF X>Y THEN LET Y=Y+1
22Ø IF X<Y THEN LET Y=Y-1
```

can be greatly speeded by the substitution of:

```
1Ø LET Y=Y+(X>Y)-(X<Y)
```

Sinclair BASIC does, indeed, allow this kind of comparison, and the single line may well appear to be more elegant programming, but a check with the timing program shows that the two line version is faster. If you do not own a Spectrum try the equivalent on your machine:

```
21Ø IF X>Y THEN Y=Y+1
22Ø IF X<Y THEN Y=Y-1
```

or

```
21Ø Y=Y+(X<Y)-(X>Y)
```

In each case, try suitable X and Y values.

MULTIPLICATION AND DIVISION

The expression C=D*Ø.5, and C=D/2 perform exactly the same calculation, but you will find that multiplication is slightly faster.

Try these suggestions to see which is faster, and keep the results for future reference:

```
2ØØ LET C=1Ø+1Ø
2ØØ LET C=D+D (where D=1Ø)
2ØØ LET C=1Ø*1Ø
2ØØ LET C=1Ø/1Ø
2ØØ LET C=1Ø+PI
2ØØ LET C=SIN 1Ø
2ØØ LET C=COS 1Ø
2ØØ LET C=TAN 1Ø
2ØØ LET C=VAL "1Ø"
2ØØ LET C=1Ø
2ØØ LET C=D (where D=1Ø)
```



```
200 PRINT AT 21,0; "TEST"
200 PRINT AT 21,0; A$ (where A$="TEST")
200 PRINT AT 21,0; 10+1000+
    500+5.5
200 PRINT AT 21,0; D+E+F+G (where
    D=10, E=1000 etc)
```

```
200 C=10+10
200 C=D+D (where D=10)
200 C=10*10
200 C=10/10
200 C=10+π
200 C=SIN (10)
200 C=COS (10)
200 C=TAN (10)
200 C=VAL ("10")
200 C=10
200 C=D (where D=10)
200 PRINT "TEST"
200 PRINT A$ (where A$="TEST")
200 PRINT 10+1000+500+5.5
200 PRINT D+E+F+G (where D=10,
    E=1000 etc)
```

```
200 C=10+10
200 C=D+D (where D=10)
200 C=10*10
200 C=10/10
200 C=10+PI
200 C=SIN 10
200 C=COS 10
200 C=TAN 10
200 C=VAL "10"
200 C=10
200 C=D (where D=10)
200 PRINT TAB(21,0)"TEST"
200 PRINT TAB(21,0)A$ (where A$"TEST")
200 PRINT TAB(21,0)10+1000+
    500+5.5
200 PRINT TAB(21,0)D+E+F+G (where
    D=10, E=1000 etc)
```

```
200 C=10+10
200 C=D+D (where D=10)
200 C=10*10
200 C=10/10
200 C=10+PI
200 C=SIN (10)
200 C=COS (10)
200 C=TAN (10)
200 C=VAL ("10")
200 C=10
200 C=D (where D=10)
200 PRINT @260, "TEST"
200 PRINT @260, A$ (where A$="TEST")
200 PRINT @260, 10+1000+500+5.5
200 PRINT @260, D+E+F+G (where
    D=10, E=1000 etc)
```

TIPS FOR SPEE

Using PROCedures is faster than GOSUBs. It is faster to pass parameters to a PROCedure than to use global variables—use PROCFAST(X,Y,Z) instead of PROCFAST

Remember FOR ... NEXT loops are faster than REPEAT ... UNTIL loops. Both are faster than IF ... THEN GOTO loops.

Start ARRAYS from subscript 0, not from 1.

When using a FOR ... NEXT loop do not put a variable after the NEXT.

Define Arrays at the start of the program.

Use INTEGER variables rather than FLOATING POINT whenever possible.

Use variables instead of numbers

Use short variable names; single letters where possible.

Put all frequently used subroutines at the beginning of the program.

Use FOR ... NEXT loops in preference to loops like:
100 LET X=X+1: IF X<20 THEN GOTO 100

Use low line numbers—start programming from line 10 rather than line 1000

Use short Machine Code routines called from Basic for Scrolling, etc. Become an avid collector of published machine code routines.

SORTING AND SEARCHING

The topic of sorting has been covered previously in *INPUT*—see pages 392 to 397—so it won't be examined in too great a depth here. The Shell-Metzner Sort described in the article would be the one to choose for any application involving the sorting of more than, say, one hundred items. It has the peculiarity that the more data it has to cope with, the faster it sorts. This sort is very much faster than the more well-known Bubble Sort, but is another example of where you will be sacrificing memory for speed.

Searching is often thought of alongside sorting. This time, instead of putting a collection of data in order, you'll be trying to retrieve a particular piece (or related pieces) of data as quickly as possible. Suppose you had a list of telephone numbers stored in your machine, and you needed the number of Albert Bodgitt and Sons, Painters and Decorators, the last thing you'd want is for the machine to take longer to find the number than it would take you to use a telephone book. You, therefore, are aiming to write the fastest possible sort routines.

The Serial Search (see listing below) simulates a person searching through a list, item by item, on paper.

DING UP BASIC

In calculation programs Define a Function to cover repetitive calculations; e.g.:
10 DEF FN A(x) = x − (INT (x/360)*360)
20 LET number = FN A (number)
Better than:
10 IF number > 360 THEN LET number = number − 360: GOTO 10

Plan your program on paper before touching the keyboard. In this way you will avoid losing speed by excessive jumping backwards and forwards, indicative of a badly planned program.

Remember Subroutines save Memory but waste Time.

Remove all superfluous common denominators; e.g. Change
10 LET X = Y/100 + Z/100 to
10 LET X = (Y + Z)/100

Avoid GOTOs whenever possible.

Use multistatement lines when possible.

Remove all unnecessary spaces, blank lines and REM statements from the program.

When employing an IF statement, put the most likely FALSE condition first.

Re-use Variable names and loop Variables, rather than defining extra ones.

Use variable names with an even spread throughout the alphabet—for example, don't have all the variable names starting with S.

If you are using integer variables as well as floating point ones, place the integer variables at the front of the line, where they will be executed first.

```
10 DATA "ANTELOPE","BEAVER",
   "DOLPHIN","FOX","GOAT",
   "IGUANA","JACKASS","MUSTANG",
   "PIGLET","SCORPION"
20 RESTORE 10: DIM B$(10,8)
30 CLS : FOR i = 1 TO 10: READ B$(i): PRINT
   AT i,5;B$(i): NEXT i
40 POKE 23658,8: INPUT "Enter
   keyword□";A$
50 FOR X = 1 TO 10
60 IF B$(x, TO LEN A$) = A$ THEN PRINT
   FLASH 1;AT X,13;""; FLASH 0;"FOUND":
   GOTO 40
70 NEXT X
```

```
10 REM SERIAL SEARCH
20 DIM B$(10)
30 PRINT "■■■■":FOR I = 1 TO
   10:READ B$(I):PRINT I,B$(I):NEXT I
40 INPUT "■ENTER KEYWORD
   ■";A$:PRINT "■■■■"
50 FOR X = 1 TO 10
60 IF B$(X) = A$ THEN PRINT
   SPC(4);"■FOUND":GOTO 75
70 PRINT:NEXT X
75 PRINT "■"
80 DATA ANTELOPE,BEAVER,
   DOLPHIN,FOX,GOAT,IGUANA
85 DATA JACKASS,MUSTANG,
   PIGLET,SCORPION
```

```
10 MODE6
20 DIM N$(10)
30 F% = FALSE
40 FOR C% = 1 TO 10:READ NA$:
   N$(C%) = NA$:NEXT
50 INPUT"ANIMAL TO BE FOUND";A$
60 FOR C = 1 TO 10:IF N$(C) = A$ THEN
   PRINTA$,C:F% = TRUE
70 NEXT
80 IF F% = FALSE THEN PRINT'''
   "NOT FOUND"
85 TIME = 0:REPEAT UNTIL TIME > 200
90 RUN
100 DATA ANTELOPE,BEAVER,
    DOLPHIN,FOX,GOAT,IGUANA,
    JACKASS,MUSTANG,PIGLET,
    SCORPION
```

```
10 REM SERIAL SEARCH
20 DIMB$(10)
30 CLS:FORI = 1TO10:READB$(I):
   PRINT@33 + I*32,B$(I);:NEXT
40 PRINT@417,STRING$(30,32);
   STRING$(30,8);:INPUT"ENTER
   KEYWORD□";A$
50 FORX = 1TO10
60 IF B$(X) = A$ THENPRINT@46
   + X*32,CHR$(191);"FOUND";:
   X = 10
70 NEXT:GOTO40
80 DATA ANTELOPE,BEAVER,
   DOLPHIN,FOX,GOAT,IGUANA,
   JACKASS,MUSTANG,PIGLET,
   SCORPION
```

Q+A

Why do the speed tips only apply to some machines?
There are no hard-and-fast rules about how to write BASIC programs to extract the final spurt of speed, and equally, there are no hard-and-fast rules about writing interpreters—after all, they are just machine programs. They can have similar compromises to any programs you may write.

The speed of an interpreter will depend on how it's written, and the features the manufacturer provides in BASIC.

Although the Serial Search is a very well-known search routine, it's not particularly quick. The Binary Search (see listing below) is not much more difficult to program, but is much faster, although you will only notice the difference when searching large lists.

The greater speed of the Binary Search is achieved because, unlike the Serial version, it does not have to examine every item on the list. The data must have been previously sorted into numeric or alphabetical order, and the computer first looks at the element in the centre of the list. From this point, it moves up or down, each time cutting the remaining list in half, as it compares each element encountered with the item it is searching for. Initially, it is not looking for an exact match, as in the Serial Search, but simply notes if the first letter is higher or lower than that of the desired item.

```
10 CLS : RESTORE
20 LET t=10: LET b=1
30 DIM n$(10,10)
40 FOR c=1 TO 10
50 READ n$(c)
60 NEXT c
70 INPUT "ANIMAL TO BE FOUND";
   a$
75 LET a$=a$+
   "□□□□□□□□□□□"
   (TO 10-LEN a$)
80 PAUSE 50
95 CLS
100 IF n$(t)=a$ THEN PRINT n$(t),t:
   GOTO 200
110 IF n$(B)=a$ THEN PRINT n$(b),b:
   GOTO 200
120 LET p=INT (0.5+(t+b)/2)
130 IF n$(p)=a$ THEN PRINT n$(p),p:
   GOTO 200
140 IF n$(p)>a$ THEN LET t=p
150 IF n$(p)<a$ THEN LET b=p
160 IF t-b=1 THEN PRINT "□□□NOT
   FOUND": GOTO 200
170 GOTO 100
200 IF INKEY$="" THEN GOTO 200
210 RUN
580 DATA "Antelope","Beaver",
   "Dolphin","Fox","Goat",
   "Iguana","Jackass","Mustang",
   "Piglet","Scorpion"
```

```
10 PRINT "♥"
20 T%=10:B%=1
30 DIM N$(10)
40 FOR C=1 TO 10
50 READ N$(C)
60 NEXT C
70 INPUT "ANIMAL TO BE FOUND
   ▮";A$
80 TI$="000000"
90 IF TI<50 THEN 90
95 PRINT "♥"
100 IF N$(T%)=A$ THEN PRINT
   N$(T%),T%:GOTO 200
110 IF N$(B%)=A$ THEN PRINT
   N$(B%),B%:GOTO 200
120 P%=(T%+B%)/2
130 IF N$(P%)=A$ THEN PRINT
   N$(P%),P%:GOTO 200
140 IF N$(P%)>A$ THEN T%=P%
150 IF N$(P%)<A$ THEN B%=P%
160 IF T%-B%=1 THEN PRINT
   "▮▮▮▮NOT FOUND":GOTO 200
170 GOTO 100
200 GET Y$:IF Y$="" THEN 200
210 RUN
580 DATA ANTELOPE,BEAVER,
   DOLPHIN,FOX,GOAT,IGUANA
585 DATA JACKASS,MUSTANG,
   PIGLET,SCORPION
```

```
10 MODE 6
20 T%=10:B%=1
30 DIM N$(10)
40 FOR C=1 TO 10
50 READ N$(C)
60 NEXT
70 INPUT"ANIMAL TO BE FOUND",
   A$
80 TIME=0:REPEAT UNTIL TIME>
   100:CLS
90 PRINT''''''''
100 IF N$(T%)=A$ THEN PRINT
   N$(T%),T%:GOTO 200
110 IF N$(B%)=A$ THEN PRINT
   N$(B%),B%:GOTO 200
120 LET P%=(T%+B%)/2
130 IF N$(P%)=A$ THEN PRINT
   N$(P%),P%:GOTO 200
140 IF N$(P%)>A$ THEN T%=P%:GOTO 160
150 B%=P%
160 IF T%-B%=1 THEN PRINT
   "□□□□NOT FOUND":GOTO 200
170 GOTO 100
200 TIME=0:REPEAT UNTIL
   TIME>200:RUN
580 DATA ANTELOPE,BEAVER,
```

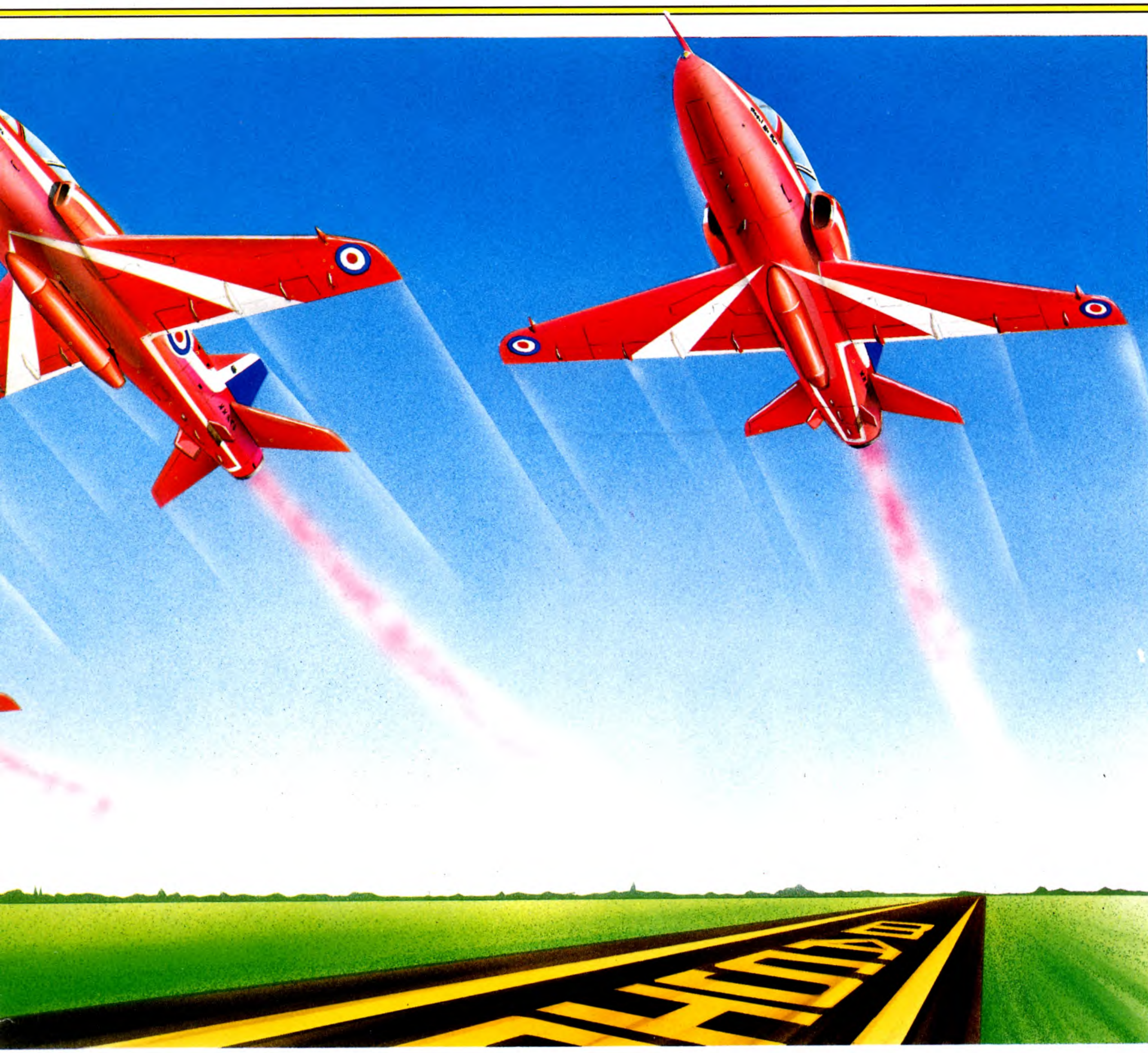
Q+A

BASIC's still too slow, but I find machine code too tedious. What else can I do?

There is a kind of 'half-way house' in languages which are compiled rather than interpreted—interpreting BASIC statements as the program is RUNning is what takes the time.

Instead, a compiled language uses another type of program—rather like the assemblers you may be familiar with from machine code programming—which converts the program from a high level language into machine code before RUNning the program. This means that after compiling, you are RUNning a program in machine code, rather than a high level language.





```
   DOLPHIN,FOX,GOAT,IGUANA,
   JACKASS,MUSTANG,PIGLET,
   SCORPION
```

```
10 CLS
20 T=10:B=1
30 DIM N$(10)
40 FORC=1TO10
50 READ N$(C)
60 NEXT
70 INPUT"ANIMAL TO BE FOUND□";
   A$
80 TIMER=0
90 IF TIMER<50 THEN 90 ELSECLS
100 IF N$(T)=A$ THENPRINT
   N$(T),T:GOTO200
110 IF N$(B)=A$ THEN PRINT
   N$(B),B:GOTO200
120 P=INT(.5+(T+B)/2)
130 IF N$(P)=A$ THEN PRINT
   N$(P),P:GOTO200
140 IF N$(P)>A$ THEN T=P
150 IF N$(P)<A$ THEN B=P
160 IF T-B=1 THEN PRINT
   "□□□□NOT FOUND":GOTO200
170 GOTO100
200 IF INKEY$="" THEN 200
210 RUN
580 DATA ANTELOPE,BEAVER
   DOLPHIN,FOX,GOAT,IGUANA,
   JACKASS,MUSTANG,PIGLET,
   SCORPION
```

There is no one answer to speeding Basic programs, the best results are achieved by careful attention to lots of apparently insignificant details, together with a great deal of experimenting. Since every program is unique there can never be any hard and fast rules about how a program should be written. There is always something new to be discovered, and here lies part of the fascination of programming. And the greatest speed gain for your program is likely to come from the discovery you make yourself.

CLIFFHANGER: ADDING INSTRUCTIONS

Once the title page has been printed—and the player knows which game he is supposed to be playing—he needs to be told how to play. When writing your own games it is easy to forget that others will not know how to play them. You should really try to get as much playing information on the screen as possible. Nothing is more irritating than having to stop and refer to printed notes when the game is in full swing.

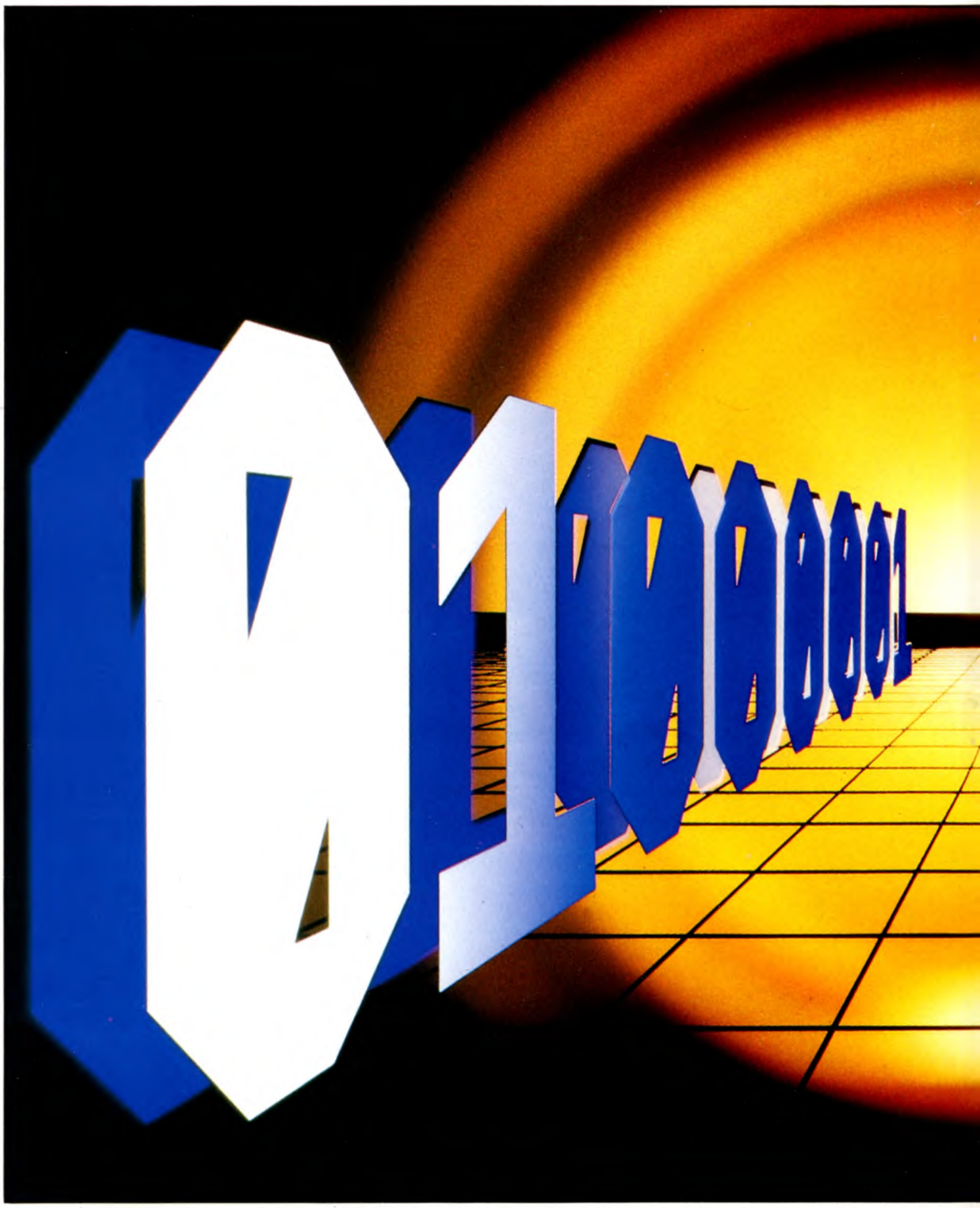
The instructions themselves should be clear and concise. But it is best to give a bit of a story line at this point too. Games are supposed to appeal to the imagination as well as the reflexes.

Again a BASIC program is used to POKE the instruction data letter by letter into an ASCII table when it is RUN.

```
10 LET x=57480
20 FOR n=1 TO 16
30 READ a$: FOR o=1 TO LEN a$: POKE x,
   CODE (a$(o TO o)): LET x=x+1: NEXT o
40 NEXT n
100 DATA "□After a short walk
   Willie□□□□□□"
120 DATA "returns to find the goats
   have□□"
140 DATA "spread his picnic goodies
   all□□□"
160 DATA "over a rocky embankment.
   □□□□□□□□"
180 DATA "Willie sets off to reclaim his□□"
200 DATA "lost possessions,but is hampered"
220 DATA "by falling boulders,pot
   holes,□□"
240 DATA "and vicious snakes.To
   make□□□□□□"
260 DATA "matters worse the tide is rising"
280 DATA "and he is in danger of being cut"
290 DATA "off.To help Willie in his quest□"
300 DATA "read the following and press 'S'"
320 DATA "to start.□□□□□□□
   □□□□□□□□□"
330 DATA "N□□□□—□Run□□□□
   □□□□□□□□□□□□□
   □□□□□"
340 DATA "M□□□□—□Vertical□
   jump□□□□□□□□□□□□"
360 DATA "Both□—□Diagonal
   □jump"
500 FOR n=57435 TO 58000
510 PRINT CHR$ PEEK n;
520 NEXT n
```

This data is picked up by the following machine code program and printed on the screen.

```
org 58104
call cl
ld ix,57480
ld hl,32
ld a,7
ld b,255
call me
ld b,138
call me
```

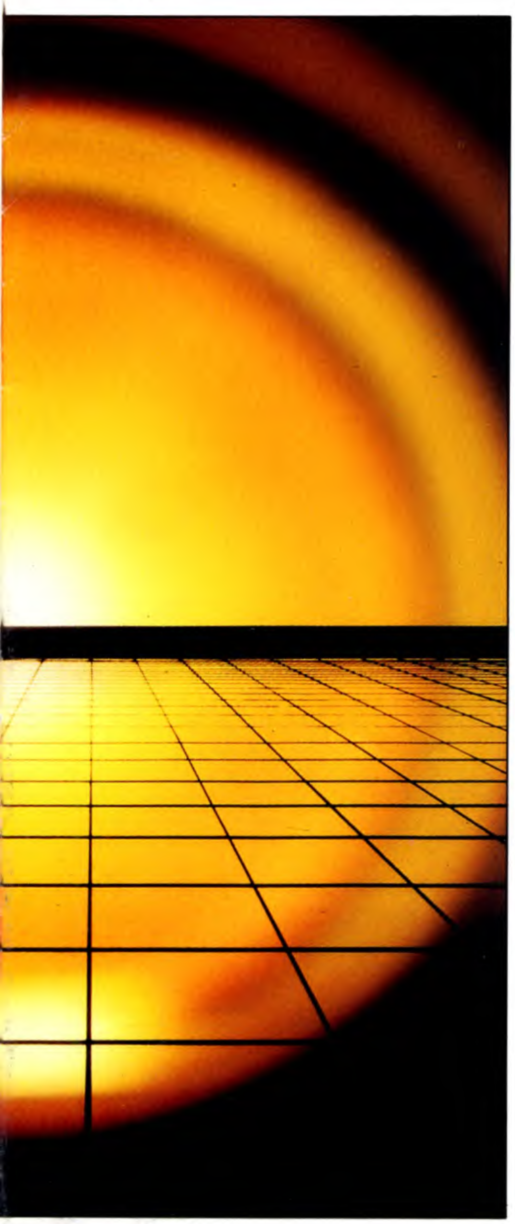


Willie will have to wait. It's no good trying to help him save his picnic before you know what to do. So first you have to print the instructions on the screen

- CONSTRUCTING ASCII DATA TABLES
- PRINTING LINES
- MAKING OPERATING SYSTEM CALLS

The machine code game 'CLIFFHANGER' which appears in this and subsequent issues of *INPUT* is not connected with, and bears no resemblance to, the game 'CLIFF HANGER' owned and distributed by NEW GENERATION SOFTWARE

```
        ld de,39
        add hl,de
        ld b,100
        ld a,70
        call me
ktt     ld a,253
        in a,254
        bit 1,a
        jr nz,ktt
        ret
```



```
        org 58192
cl      *
        org 58155
me      *
```

These two programs should be assembled and SAVEd in the same way as the two programs in part one of Cliffhanger. When you call this routine, the routines given in part one of Cliffhanger must also be in memory so they can be called.

PRINTING THE INSTRUCTIONS

The **cl** routine is called again to clear the title page off the screen. The **me** routine is used again to print on the screen. But the registers first have to be loaded with the appropriate parameters.

The start address of the new data table is loaded into IX, the screen position of the beginning of the line to be printed into HL, the colour into A and the string length into B. And the **me** routine is called again, repeatedly, to print the instructions.

The **me** routine is then called twice, first with 255 in B and then with 138 in B. The routine has to be called twice because 255 is the maximum an eight-bit register like B can hold. The 138 takes in the remainder of the string data. A new screen position does not have to be loaded into A for the second part of the string data as it follows the first.

The third part of the data is given a new screen position though. A gap is needed between the end of the text and the keypress instructions. This can be done either by putting a row of spaces in the data string, or by giving HL a new value. Here, both methods are illustrated. There are 16 spaces in the data at this point and in the assembly language program 39 is put into DE, then DE is added to HL. The HL register pair acts as a 16-bit accumulator and the result of the addition stays in HL, which is where you need the screen position parameter when the **me** routine is called.

The rest of the string is 100 characters long and is in bright yellow—colour 70—rather than white—colour 7—like the main body of the text.

WAITING TO START

The instructions stay on the screen until you press S to start the game. To do this, a machine code routine using the **in** command is used.

The **in** is used to look at the keyboard in exactly the same way as on page 482, where it was looking for the [BREAK] and [SYMBOL SHIFT] keys to be depressed. Here though, you are looking for an S. The 253 in A directs the **in** to that part of the keyboard which is accessed, as before, through port 254.

This time, bit 1 being set means that the S is being pressed. So **bit 1,a** looks at bit 1. This sets the zero flag if bit 1 of the accumulator is set.

The processor simply goes round and round the **ktt** loop until S is pressed, bit 1 is set and the **jr nz** condition is not set. The processor would normally go onto the rest of the program then. But for now it hits **ret** and returns to BASIC as this is the end of part two of Cliffhanger.

This program prints up the instructions after the title page when the program in part one of Cliffhanger is called. The **ret** which ended the first part is overwritten when you assemble this part.

Again a BASIC program is used to POKE the instruction data letter by letter into an ASCII table when it is RUN.

```
10 AD=17312:I=0
15 POKEAD+I,147:I=I+1
50 N=3:GOSUB2000
60 POKEAD+I,31:I=I+1
70 A$="THE STORY SO FAR□.□.
   □.□.":N=24:GOSUB1000
80 N=1:GOSUB2000
90 A$="□□□□□□□□□□□□□□□□□□□□":
   N=17:GOSUB1000
100 N=3:GOSUB2000
110 A$="□□□□□□AFTER A SHORT
   WALK WILLY RETURNS□□":
   N=40:GOSUB1000
120 A$="TO FIND THE GOATS HAVE
   SPREAD HIS PICNIC":N=40:GOSUB1000
```

```
130 A$="GOODIES ALL OVER A ROCKY
    EMBANKMENT.□□□□":N=4Ø:
    GOSUB1ØØØ
140 A$="□□□□□□□WILLY SETS
    OFF TO RECLAIM HIS LOST":
    N=4Ø:GOSUB1ØØØ
150 A$="POSSESSIONS,□BUT IS
    HAMPERED BY FALLING□□":
    N=4Ø:GOSUB1ØØØ
160 A$="BOULDERS,□POTHOLES AND
    VICIOUS SNAKES.□□":N=4Ø:
    GOSUB1ØØØ
170 A$="TO MAKE MATTERS WORSE
    THE TIDE IS RISING":N=4Ø:
    GOSUB1ØØØ
180 A$="AND HE CANNOT SWIM.
    □□□□□□□□□□□□□
    □□□□□□□□□□":N=4Ø:
    GOSUB1ØØØ
190 A$="□□□□□□□TO HELP
    WILLY ON HIS QUEST READ□□□":
    N=4Ø:GOSUB1ØØØ
200 A$="THE FOLLOWING CONTROL
    INSTRUCTIONS AND□□":N=4Ø:
    GOSUB1ØØØ
210 A$="PRESS□□S□□TO START.":
    N=19:GOSUB1ØØØ
220 N=3:GOSUB2ØØØ
230 POKEAD+I,149:I=I+1
240 A$="□□□□□KEYBOARD□ – □
    SHIFT=MOVE RIGHT□□□□□□
    □□":N=4Ø:GOSUB1ØØØ
250 A$="□□□□□□□□□□□
    □□□□□□□□Z□□□=JUMP
    VERTICAL□□□□□":N=4Ø:
    GOSUB1ØØØ
260 A$="□□□□□□□□□□□□
    □□□□□BOTH□=JUMP RIGHT
    □□□□□□□□□":N=4Ø:
    GOSUB1ØØØ
270 N=1:GOSUB2ØØØ
280 A$="□□□□□JOYSTICK□ – □
    INSERT IN PORT 2□□□□□□□
    □□":N=4Ø:GOSUB1ØØØ
290 POKEAD+I,Ø
300 STOP
1000 REM INSTRUCTION TEXT POKER
    SUBROUTINE
1010 FOR J=1TON
1020 POKEAD+I,ASC(MID$(A$,J,1))
1030 I=I+1:NEXT:RETURN
2000 REM INSERT RETURN SUBROUTINE
2010 FOR J=1TON:POKEAD+I,13:
    I=I+1:NEXT:RETURN
```

This data is picked up by the following machine code program and printed on the screen.

```
ORG  16480
LDA  #$Ø9
STA  $DØ2Ø
LDA  #$ØF
STA  $DØ21
LDA  #$AØ
STA  $FB
LDA  #$43
STA  $FC
LDY  #$ØØ
NOP
LDA  ($FB),Y
BEQ  $4Ø87
JSR  $FFD2
NOP
INC  $FB
BNE  $4Ø83
INC  $FC
CLC
BCC  $4Ø75
NOP
JSR  $FFE4
CMP  #$53
BNE  $4Ø87
RTS
```

These two programs should be assembled and SAVEd in the same way as the two programs in part one of Cliffhanger.

THE BASIC

Following the two methods used to supply the data for the words printed on the title page, the BASIC program here uses a third way to construct an ASCII table. This time the words are put in as strings, the ASCII of each letter is POKEd into the table by the subroutine starting at Line 1ØØØ.

But first of all the screen has to be cleared. So Line 15 POKEs 147—the ASCII for [CLR/HOME]—into the table. Then Line 5Ø puts three carriage returns in the table for formatting purposes, and Line 6Ø POKEs in the code for blue ink into the table.

Line 29Ø POKEs Ø into the last location in the table, so that the machine code program knows when the time has come for it to come to the end.

THE ASSEMBLY LANGUAGE

The number 9 is stored in DØ2Ø, which sets the border colour to brown. Then the screen colour—which is controlled by memory location DØ21—is set to F or grey.

The start address of the new data is then stored in zero-page location FB and FC, and the index register Y is set to Ø.

LDA ($FB),Y loads up the data table a byte at a time using indirect addressing. BEQ $4Ø87 exits the routine when a zero is loaded into the accumulator—in other words, the processor moves onto the next part of the program when the end of the table is reached.

Again the ROM subroutine at FFD2 is used to print the character whose ASCII is in the accumulator on the screen.

The next little routine increments the pointer in FB and FC. BNE $4Ø83 branches over the instruction which increments the high byte of the pointer. The pointer is incremented while the Y index register is held to zero because the data table is more than 255 bytes long.

The carry flag is then cleared and the processor branches back to load up the next byte of the table.

WAITING FOR THE OFF

The ROM routine at FFE4 is the GETIN routine which gets a character from the queue in the keyboard buffer. The ASCII of the character is returned in the accumulator and CMP #$53 compares it with 53 hex or 83 decimal, which is the ASCII for S.

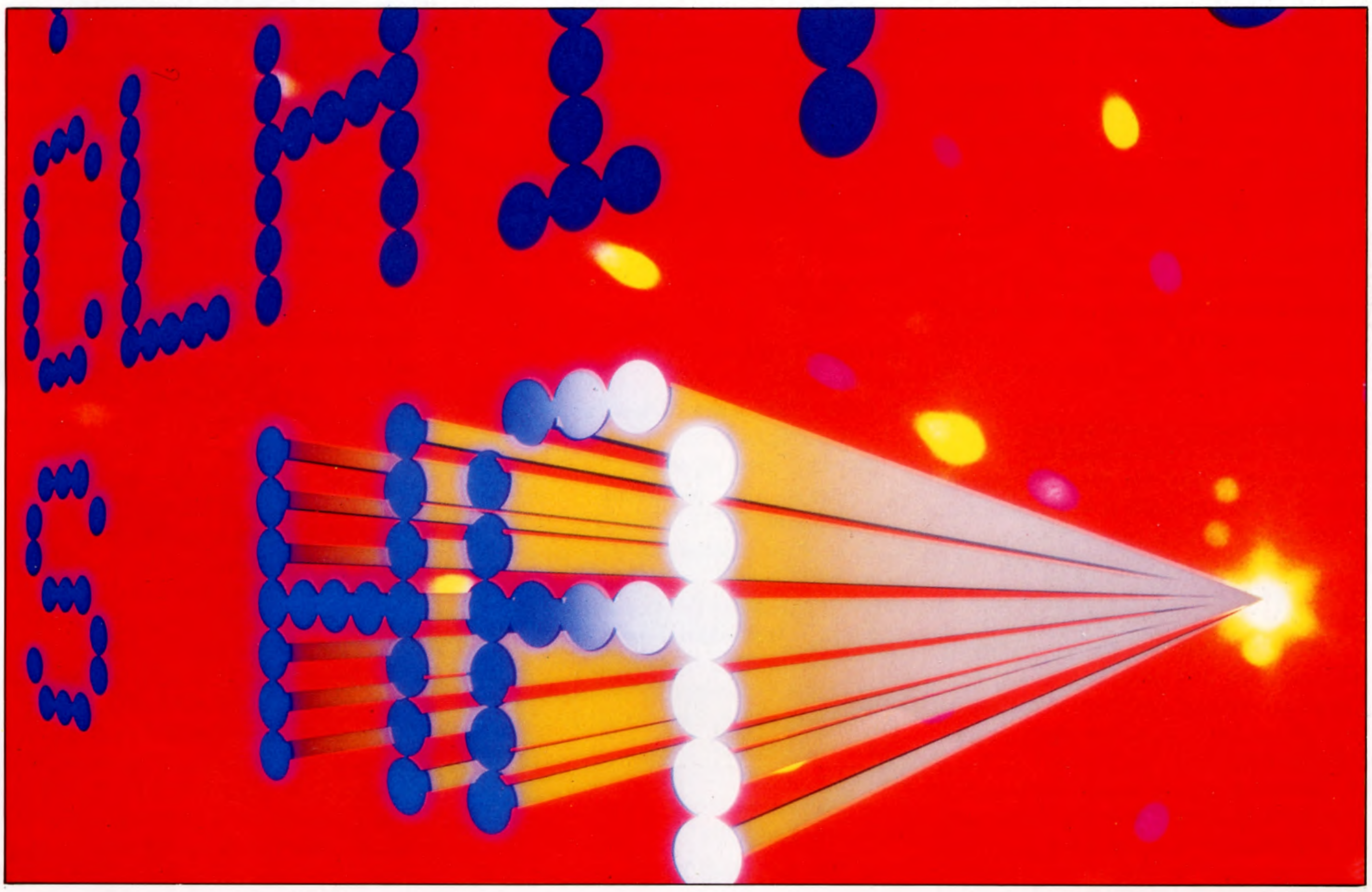
If an S has not been pressed and the ASCII character does not appear in the keyboard buffer, the BNE $4Ø87 sends the processor back round this little loop to check the next byte in the keyboard buffer.

But if an S has been pressed, the BNE condition is not fulfilled and the processor moves onto the next routine. Only, in this case it BReaKs because this is the end of part two of Cliffhanger.

The BASIC part of the following program puts the instruction data into a data table in the protected part of memory where the machine code routine can access it, and the assembly language prints the instructions on the screen. Press [BREAK] and type PAGE= &3ØØØ then NEW before typing it in.

```
80 DATA23,Ø,1Ø,32,Ø,Ø,Ø,Ø,Ø,Ø
90 FOR A%=&DØ2TO&DØB:READ?
    A%:NEXT
140 P%=&11D6:FOR B%=1 TO 16
150 READ ?P%,?(P%+1),A$:
    $(P%+2)=A$:P%=P%+3+LENA$:
    NEXT
160 ?P%=13:IF P%>&13FE THEN PRINT
    "TOO MUCH DATA DON'T CALL MACHINE
    CODE":END
170 DATA6,1,BBC Cliffhanger
    By D. Summers
180 DATA6,2,**************************
190 DATA4,4,After a short walk
    Willie returns to
200 DATAØ,5,find the goats have
    spread his picnic
210 DATAØ,6,goodies all over a rocky
    embankment.
220 DATAØ,7,Willie sets off to reclaim his lost
230 DATAØ,8,"possessions,but finds he is
    hampered by"
240 DATAØ,9,"falling boulders, potholes and
    vicious"
250 DATAØ,1Ø,snakes.To make matters worse the
    tide is
260 DATAØ,11,rising and he is in danger of being
    cut
270 DATAØ,12,off. To help Willie in his quest read
    the
280 DATAØ,13,following instructions:—
290 DATA16,16,N – Left,16,18,M – Right, 16,2Ø
```

```
,SHIFT—Jump,8,23, Press SPACE for start
3ØØ ?&13F7=13
34Ø FORPASS=ØTO3STEP3
35Ø P%=&13F8
36Ø [OPTPASS
37Ø .Inst
38Ø LDA#22
39Ø JSR&FFEE
4ØØ LDA#6
41Ø JSR&FFEE
42Ø JSRLb11
43Ø LDA#&D6
44Ø STA&7Ø
45Ø LDA#&11
46Ø STA&71
47Ø .Lb1
48Ø LDX#4Ø
49Ø .Lb2
5ØØ DEX
51Ø LDA#31
52Ø JSR&FFEE
53Ø TXA
54Ø JSR&FFEE
55Ø LDY#1
56Ø LDA(&7Ø),Y
57Ø JSR&FFEE
58Ø .Lb3
59Ø INY
6ØØ LDA(&7Ø),Y
61Ø CMP#13
62Ø BEQLb4
63Ø JSR&FFEE
64Ø STX&72
65Ø TYA
66Ø CLC
67Ø ADC&72
68Ø CMP#41
69Ø BNELb3
7ØØ .Lb7
71Ø LDA#8Ø
72Ø JSRLb8
73Ø STY&72
74Ø LDY#Ø
75Ø TXA
76Ø CMP(&7Ø),Y
77Ø BNELb2
78Ø LDY&72
79Ø INY
8ØØ INY
81Ø TYA
82Ø CLC
83Ø ADC&7Ø
84Ø STA&7Ø
85Ø BCCLb5
86Ø INC&71
87Ø .Lb5
88Ø LDY#Ø
89Ø LDA(&7Ø),Y
9ØØ CMP#13
91Ø BNELb1
92Ø .Lb6
93Ø LDA#&81
94Ø LDY#&FF
95Ø LDX#&9D
96Ø JSR&FFF4
97Ø TXA
98Ø BEQLb6
99Ø RTS
1ØØØ .Lb4
1Ø1Ø LDA#32
1Ø2Ø JSR&FFEE
1Ø3Ø DEY
1Ø4Ø JMPLb7
1Ø5Ø .Lb8
1Ø6Ø STA&72
1Ø7Ø TXA
1Ø8Ø PHA
1Ø9Ø TYA
11ØØ PHA
111Ø LDX&72
112Ø .Lb9
113Ø LDY#&FF
114Ø .Lb1Ø
115Ø DEY
116Ø BNELb1Ø
117Ø DEX
118Ø BNELb9
```



```
1190 PLA
1200 TAY
1210 PLA
1220 TAX
1230 RTS
1240 .Lb11
1250 LDX#2
1260 .Lb12
1270 LDA&D00,X
1280 JSR&FFEE
1290 INX
1300 CPX#&C
1310 BNELb12
1320 RTS
1330 ]NEXT
```

Save the source code, then RUN this program to construct the data table and assemble the machine code. To execute it use:

```
CALL &13F8
```

When you SAVE this program, *SAVE only from &11D6 onwards to &1492.

THE DATA

Lines 80 and 90 contain the DATA which switch off the cursor. This DATA was given before in Lines 80 and 90 of part one of Cliffhanger. But it is given again here so that you can test this program independently.

There is no need to save the part of the machine code data table this DATA is put in though. When the machine code program given in part one is in memory as well, part two can access its data table. The instruction data is in Lines 140 to 300. Memory locations are simply defined as strings. These are put byte by byte into the memory locations defined and the ones following it. The instructions themselves are in plain English and the two character strings at the beginning of each line define the X and Y coordinates of where that line of text should start on the screen.

THE ASSEMBLY LANGUAGE

Once the assembler has been set up, the instructions in Lines 380 to 410 switch your computer into MODE 6. Line 420 jumps to the subroutine in Lines 1240 to 1320 which pick up the data in Line 80 and switch the cursor off.

Lines 420 to 460 store the start address of the instruction data table in &70 and &71. The actual printing on the screen is done by three nested loops. Loop one starts on Line 470 and counts down the lines of text. Loop two starts on 490 and runs each line of text on from the right—this is why X is loaded with 40, the character position on the extreme right-hand side of the screen (once it has been decremented by the instruction in Line 500), in Line 480. And loop three starts on Line 580 and prints each character at the right of the screen while the rest of the line is shifted to the left.

Lines 510 and 520 are the machine code equivalent of the VDU 31 which enables the text cursor to be moved to the X and Y coordinates specified in Lines 530 to 570. First the X coordinate is transferred from the X register into A, then the Y register is used to index the data from the second of CHR$s in each line and put it in A. The 'output character to screen routine' at &FFEE is called each time.

Line 590 increments Y so Line 600 picks up the next character—the first character of text on the first pass. And Line 610 checks for a 13—a return. If it finds one it doesn't bother with the rest of the line of text and sends it off to the routine which prints spaces starting at Line 1000. The instruction in Line 630 actually prints the character.

Loop three is closed by Line 690. Line 720 calls a pause routine—to give you time to read the line—starting at Line 1050. Loop two is closed by Line 770 and loop one is closed by 910. Line 900 checks for a second 13, the end of data marker POKEd in by Line 300.

Lines 920 to 980 wait for the space bar to be pressed to continue by calling the OSBYTE routine at &FF4. (OSBYTEs and the other operating system routines will be explained in a later article.)

Again a BASIC program is used to POKE the instruction data letter by letter into an ASCII table. Add the following to the BASIC program in part one of Cliffhanger and RUN it:

```
20 FORI=1 TO 4
70 NEXT A,I
85 DATA"□after a short walk willie□□□
   □□□returns to find the goats
   have□□spread his picnic
   goodies all□□□over a rocky
   embankment. □□□□□□□□□□
   willie sets off to reclaim his lost possessions,
   but is□"
90 DATA "hamperedby falling boulders,pot□
   holes,□□and vicious snakes. to make□□
   □□□□matters worse the tide is risingand
   he is in danger of being cutoff.to help willie
   in his quest read the following and press 's'to
   start.n□□□□ – □run"
100 DATA "m□□□□
   – □vertical jumpboth
   – □diagonal jump"
```

This data is picked up by the following machine code program and printed on the screen.

```
        ORG 19062
        LDX #1024
        LDY #17060
        CLRB
        JSR LPRINT
        LDB #137
        JSR LPRINT
        LEAX 24,X
        LDB #10
        JSR LPRINT
        LEAX 22,X
        LDB #20
        JSR LPRINT
        LEAX 12,X
        LDB #20
        JSR LPRINT
KEY     JSR 32774
        CMPA #83
        BNE KEY
        RTS
LPRINT  EQU 19174
```

These two programs should be assembled and SAVEd in the same way as the two programs in part one of Cliffhanger. When you call this routine, the routines given in part one of Cliffhanger must be in memory.

PRINTING THE INSTRUCTIONS

The control routine uses the data table and the same LPRINT routine that the title page used to print the instructions on the screen. Again the X register is loaded with the first print position on each line and the B register carries the length of the string to be printed on that line. And once the Y register which carries the data table pointer has been moved to the beginning of the appropriate section it moves down the table, line by line, by itself.

The LEAX instructions increment the print position to the beginning of each new line—that's why there is no space between the words that make up the end of one line and the beginning of the next in the data table.

The CLRB instruction clears the B register. This is a quick way of loading it with zero, which acts like 256 when decremented.

The KEY routine waits for you to press the S key to start the game. The first thing it does is jump to the ROM routine at 32,774 which checks to see whether a key has been pressed. On the Tandy it is at 41,409, so the hold instruction should read JSR 41409.

If a key has been pressed the value of the character is returned in the accumulator. This is compared to 83, the ASCII for S. If S has not been pressed the microprocessor simply goes round and round the KEY loop checking for a keypress until S is pressed.

ENGINEERING A SOLUTION

- THE SIMPLEST MACHINE
- MECHANICAL ADVANTAGE
- REACHING GREATER HEIGHTS
- SAVING EFFORT
- THE HYDRAULIC RAM

Mechanics is not just for engineers. It is the study of interacting forces, and appears in all sorts of everyday activities which can be analyzed by your computer

The construction of many of the world's great monuments—such as the pyramids of Egypt, and Britain's Stonehenge—is remarkable, not least because they were built long before the development of those technological devices we consider necessary for such tasks.

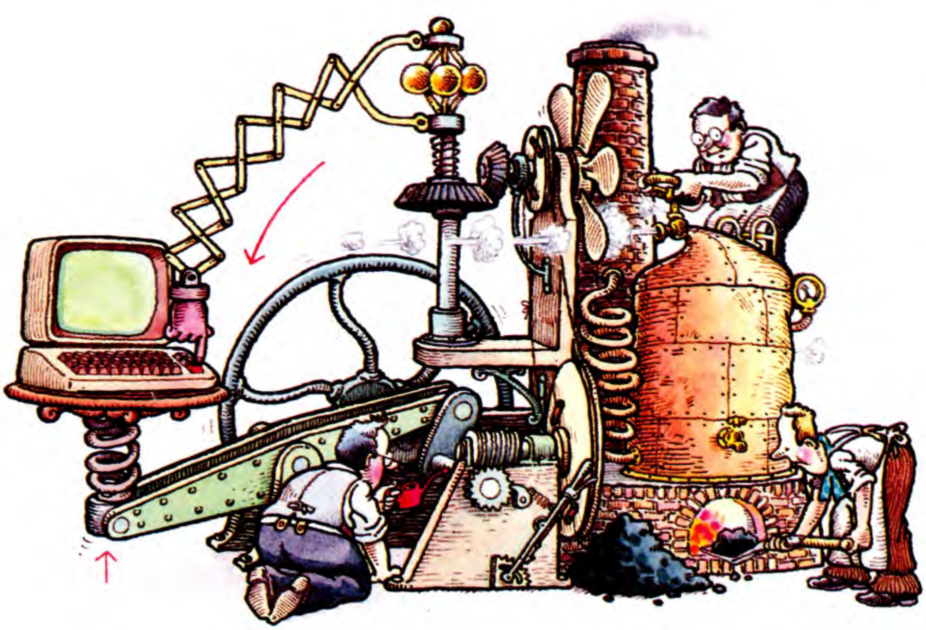
Today, few contractors would contemplate similar tasks without machinery for cutting and transporting the huge stones or for hoisting masses of material high above ground. Yet, even the most sophisticated of such equipment relies on basic principles that primitive engineers would have known and used. Chief among these is the principle of mechanics—the science of forces.

Whether a system is stationary (as a building) or in motion (as the components of a machine), there are forces at work. You can assess these instinctively, or from experience—that is what those early builders did, and what everyone does in all sorts of everyday activities. For example, if you reach to pick up a cup of coffee, you automatically apply the right force to raise it without the contents flying everywhere. And if you put up a shelf, you have a fairly good idea of what sort of timber to use, and how to support it to stop it bending under the amount of weight you put on it.

Estimating forces like this is all very well for such everyday applications, but there are many occasions when greater precision is required. And it is in this detailed analysis that the use of computers comes into its own.

You can, of course, use estimates based on experience to program computer simulations of systems involving forces. At its simplest level, this comes into the sort of simulation used in many games programs. So you can estimate, for example, how far away a sword lands when it is struck from the hands of an opponent, or how loud a noise results from two impacting objects.

But when the computer is used as a tool, you invariably need greater precision. For example, whether you are an engineer designing a bridge or a home mechanic waiting to use a hoist to raise a car engine, it is important to know when breaking point of the structure is reached. If you use your computer to analyze the forces in a structure under varying load, you need absolute precision, as well as an understanding of how to calculate forces.



THE SIMPLEST MACHINE

The problem of how to move heavy loads becomes more challenging as larger and heavier structures are built. During the 1960s, the mighty Saturn V rocket which sent men to the moon was carried on the world's largest tracked platform. Today, oil platforms destined for the North Sea are suspended on hovercraft-type pads or lubricating plastic and towed to the beach, then floated out to sea. These, and similar techniques, depend on machines of some sort—to provide pressure, traction or lift, for example. These machines in turn are merely arrangements of a few basic devices that have been used for centuries.

Among the most primitive of these is the lever. In its simplest form, a lever is a pole or rod, one end of which is placed under an object while the other end is raised to move the object. Using this device, a person can move a load several times his or her own weight. In fact, Archimedes, the great mathematician of the ancient world is credited to have claimed that he could move the Earth, if he had a long enough pole and a fulcrum. A fulcrum is the point about which a lever pivots, so in the example above, it is a place on the ground on which the pole rests.

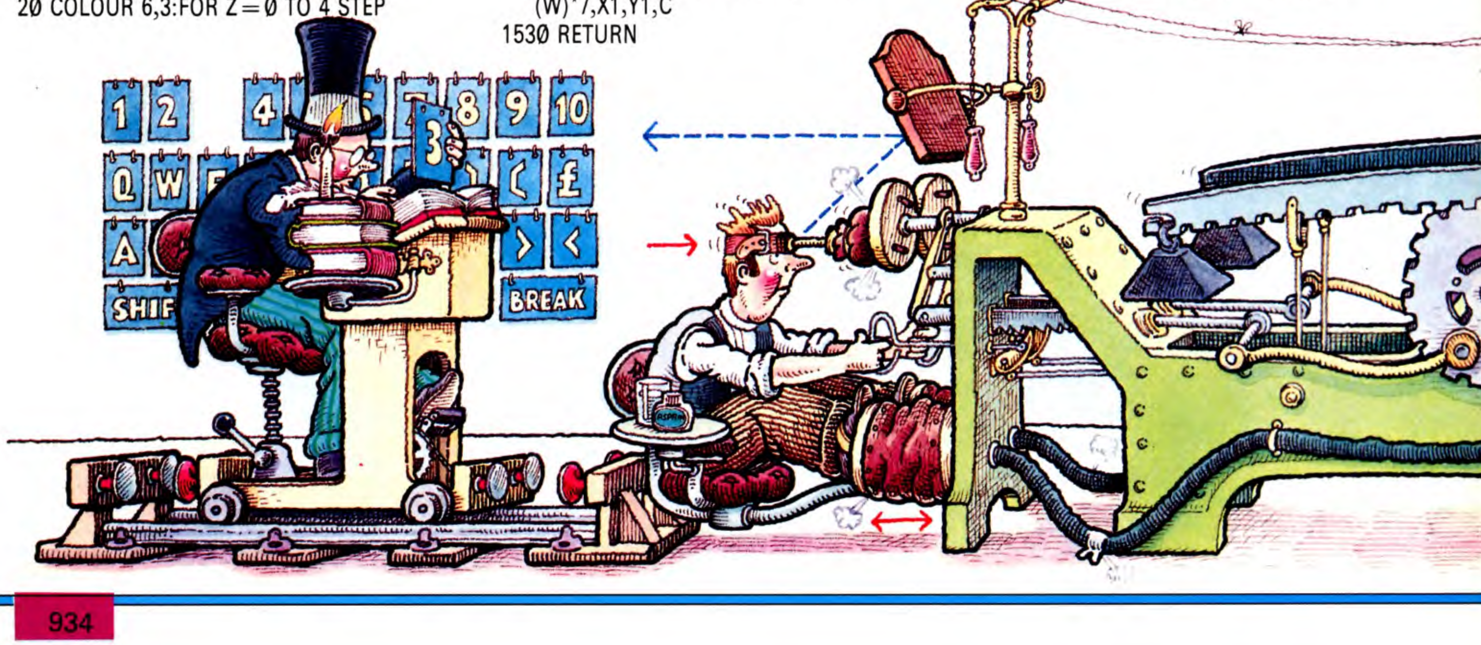
The most usual arrangement of the lever has the fulcrum not at one end of the pole or rod, but instead somewhere between both ends. Enter the first program to see this demonstrated. In order to display the hi-res graphics in the first and third programs, the Commodore 64 needs either to be fitted with a Simon's BASIC cartridge or programmed with the *INPUT* machine code hi-res utility.

```
30 BORDER Ø: PAPER Ø: INK 7: CLS : OVER 1
40 INPUT " Distance of fulcrum from
   left□"'"□□□□□□□□(1–8 mtrs
   )";d
50 IF D<1 OR d>8 THEN GOTO 40
```

```
60 LET w=(10-d)/d
70 PRINT AT 1,0;"□□Weight required to
   balance□□□□□□□□100
   KG=";w*100;" KG"
100 PLOT INK 4;0,15: DRAW INK 4;255,0:
   FOR n=0 TO 55: PLOT
   (28+20*d)-n/3,71-n
110 DRAW INK 7;2*((28+20*d)-
   PEEK 23677),0: NEXT n
120 LET a=d-10: LET a=ATN
   (a/(1-a*a))+2*ATN (1)
130 FOR b=a TO 2*ATN (1) STEP (2*ATN
   (1)-a)/10
140 FOR k=1 TO 2: GOSUB 1000: GOSUB
   1500
160 NEXT k
170 NEXT b
180 LET B=2*ATN (1): GOSUB 1000:
   GOSUB 1500
190 IF INKEY$="" THEN GOTO 190
200 RUN
1000 PLOT 128-100*SIN (b),71-
   20*d*COS (b)
1011 DRAW 127+100*SIN (b)-PEEK
   23677,71+(200-20*d)*COS (b)-PEEK
   23678
1025 DRAW 0,-8: DRAW -10,0:
   DRAW 0,-10: DRAW 20,0:
   DRAW 0,10: DRAW -9,0:
   POKE 23678,(PEEK 23678)+6
1030 RETURN
1500 PLOT 128-100*SIN (b),70-
   20*d*COS (b)
1510 LET e=SQR (SQR w)
1520 DRAW 0,-8: DRAW -10*e,0: DRAW
   0,-10*e: DRAW 20*e,0: DRAW 0,10*e:
   DRAW -9*e,0
1530 RETURN
```

```
10 HIRES 0,3
20 COLOUR 6,3:FOR Z=0 TO 4 STEP
   .5:LINE 0,180+Z*Z,319,180+Z*
   Z,1:NEXT Z
40 CSET(0):INPUT "■DISTANCE OF
   FULCRUM FROM LEFT (1-8)";D
50 IF D<1 OR D>8 THEN 40
60 W=(10-D)/D
70 PRINT"■WEIGHT REQUIRED TO
   BALANCE 100 KG=",W*100;"KG"
80 FOR G=1 TO 2000:
   NEXT G:CSET(2)
100 LINE 61+20*D,150,51+20*D,
   179,1:LINE 61+20*D,150,71+
   20*D,179,1
105 PAINT 61+20*D,160,1
120 A=10-D:A=ATN(A*A-1)
130 FOR AN=A TO 2*ATN(1) STEP
   (2*ATN(1)-A)/10
140 C=1:GOSUB 1000:C=1:
   GOSUB 1500
150 FOR G=1 TO 500:NEXT G
160 C=0:GOSUB 1000:
   GOSUB 1500
170 NEXT AN
180 AN=2*ATN(1):C=1:GOSUB 1000:
   GOSUB 1500
190 GET A$:IF A$="" THEN 190
200 RUN
1000 X1=160-100*SIN(AN):Y1=149
   -20*D*COS(AN)
1010 LINE X1,Y1,167+
   100*SIN(AN),149+
   (200-20*D)*COS(AN),C
1020 X1=160+100*SIN(AN):Y1=141
   +(200-20*D)*COS(AN)
1025 BLOCK X1,Y1,166+100*SIN
   (AN),149+(200-20*D)*COS(AN),C
1030 RETURN
1500 X1=165-100*SIN(AN):Y1=148
   -20*D*COS(AN)
1510 BLOCK 165-100*SIN(AN)-SQR
   (W)*7,148-20*D*COS(AN)-SQR
   (W)*7,X1,Y1,C
1530 RETURN
```

```
10 MODE1
20 VDU19,1,2;0;:GCOL0,129
30 CLS
40 INPUT TAB(0,1)"DISTANCE OF FULCRUM
   FROM LEFT (1-8 M)□"D
50 IF D<1 OR D>8 THEN 30
60 W=(10-D)/D
65 @%=&20209
70 PRINT''"WEIGHT TO BALANCE 100
   KG=□";W*100;" KG":@%=10
80 G=INKEY(500)
90 GCOL0,129:CLG
100 GCOL0,3:MOVE 140+100*D,205:
      MOVE 90+100*D,60:PLOT85,
   190+100*D,60
120 A=10-D:A=ATN(A*A-1)
130 FOR AN=A TO PI/2 STEP
   (PI/2-A)/10
140 GCOL0,2:PROCPOLE:GCOL0,3:
   PROCWEIGHT
160 GCOL0,1:PROCPOLE:PROCWEIGHT
170 NEXT
180 AN=PI/2:GCOL0,2:PROCPOLE:
   GCOL0,3:PROCWEIGHT
185 PRINT" ANY KEY TO CONTINUE"
190 G=GET
200 RUN
1000 DEF PROCPOLE
1010 MOVE 640-500*SIN AN,210+100*
   D*COS AN:DRAW640+500*SIN AN,
   210-(1000-100*D)*COS AN
1030 PLOT1,-10,0:PLOT0,25,0:PLOT
   0,-50,0:PLOT81,50,-30:PLOT81,
   -50,0
1040 ENDPROC
1500 DEFPROCWEIGHT
1510 MOVE 660-500*SIN AN,215+100*
   D*COS AN:PLOT0,0,SQR(W)*35:
   PLOT81,-SQR(W)*35,-SQR(W)*
```



```
   35:PLOT81,Ø,SQR(W)*35
1530 ENDPROC
```

```
10 PMODE3,1:PCLS
20 COLOR3:LINE(Ø,18Ø)-(255,191),
   PSET,BF
30 CLS
40 INPUT" DISTANCE OF FULCRUM FROM
   LEFT□□□(1-8 M)□";D
50 IF D<1 OR D>8 THEN 30
60 W=(10-D)/D
70 PRINT:PRINT"WEIGHT REQUIRED
   TO BALANCE□□□□□□□100 KG
   =";W*100;"KG"
80 FORG=1TO4000:NEXT
90 SCREEN1,Ø
100 COLOR2:LINE(28+2Ø*D,15Ø)
   -(18+2Ø*D,179),PSET
110 LINE-(38+2Ø*D,179),PSET:
   LINE-(28+2Ø*D,15Ø),PSET:
   PAINT(28+2Ø*D,16Ø),2
120 A=1Ø-D:A=ATN(A*A-1)
130 FOR AN=A TO 2*ATN(1) STEP
   (2*ATN(1)-A)/1Ø
140 C=4:GOSUB1000:C=3:GOSUB1500
150 FORG=1TO500:NEXT
160 C=1:GOSUB1000:GOSUB1500
170 NEXT
180 AN=2*ATN(1):C=4:GOSUB1000:
   C=3:GOSUB1500
190 IFINKEY$="" THEN190
200 RUN
1000 COLORC:LINE(128-1ØØ*SIN(AN),
   149-2Ø*D*COS(AN))-(134+1ØØ*
   SIN(AN),149+(2ØØ-2Ø*D)*COS
   (AN)),PSET
1020 DRAW"D2L6D6R10U6L8"
1030 RETURN
1500 COLORC:LINE(132-1ØØ*SIN
   (AN),147-2Ø*D*COS(AN))-(132-
   1ØØ*SIN(AN)-SQR(W)*7,147-2Ø*
   D*COS(AN)-SQR(W)*7),PSET,BF
1530 RETURN
```

RUN the program to see a prompt (Line 4Ø) on the screen for you to specify the distance (D) of a fulcrum from the lefthand end of a ten metre pole. Line 6Ø then calculates the weight (W) required at this end to balance a 100 kg load at the other end, and Line 7Ø prints the value of W on the screen. Lines 1ØØ to 11Ø draw the fulcrum. Line 12Ø sets up variables for the angle of the pole, and Lines 13Ø to 17Ø animate the balancing of the pole, calling a routine (Lines 1ØØØ to 1Ø3Ø) to draw the pole and another (Lines 15ØØ to 153Ø) to draw the weights. Line 18Ø draws the final position of the pole and weights.

Enter different values for D each time you RUN the program, and notice that the greatest effort (value of W) is required when the fulcrum is farthest from the 100 kg load. Conversely, when the fulcrum is nearest the load, the lever gives greatest purchase, so the effort is small.

MECHANICAL ADVANTAGE

There is a simple mathematical formula for calculating the lengths and weights in this arrangement, and it works well whenever you consider the loading on any sort of beam or pole—provided you know the position of the point about which the load acts. This formula states that the effort times its distance from the fulcrum equals the load times its distance from the fulcrum. In terms of variables, this can be written:

$$E \times DE = L \times DL$$

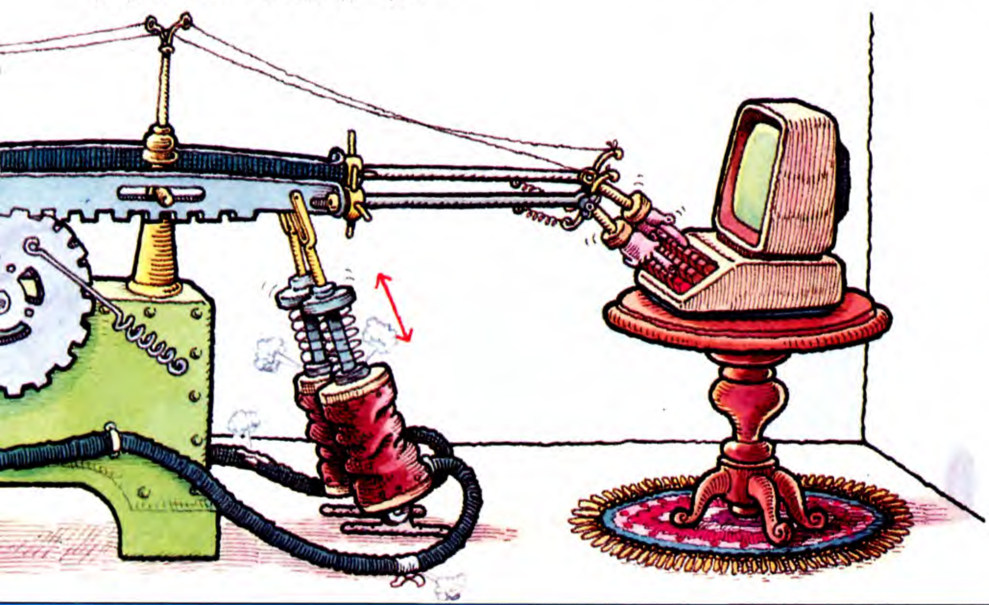
In the program above, W is used instead of E; the total length of the pole is 1Ø, so DL is 1Ø – DE. If L has a value of unity, the formula becomes

$$W = (1Ø - DE)/DE$$

which is the form used at Line 6Ø in the program above. You are asked to specify DE (D in the program), so the effort can be calculated. Whatever form of the formula you use, you can calculate the remaining variable if the others are known.

This very fact is exploited in the principle of the beam-balance weighing scales. This consists of a metal beam pivoted on a knife edge at its centre. This means that DE = DL, so they can be cancelled in the formula, leaving E = L. Now to weigh an item you simply place it on one end of the beam and place known weights on the other end until the beam balances. This is fine if you wish to weigh small quantities, as in a greengrocer's shop, but what happens when the item to be weighed is a commercial vehicle, such as a lorry laden with ore or coal? In fact, the same principle holds well, but instead of pivoting the beam at its centre, you place the knife edge near the load, so only small weights are needed to balance the beam.

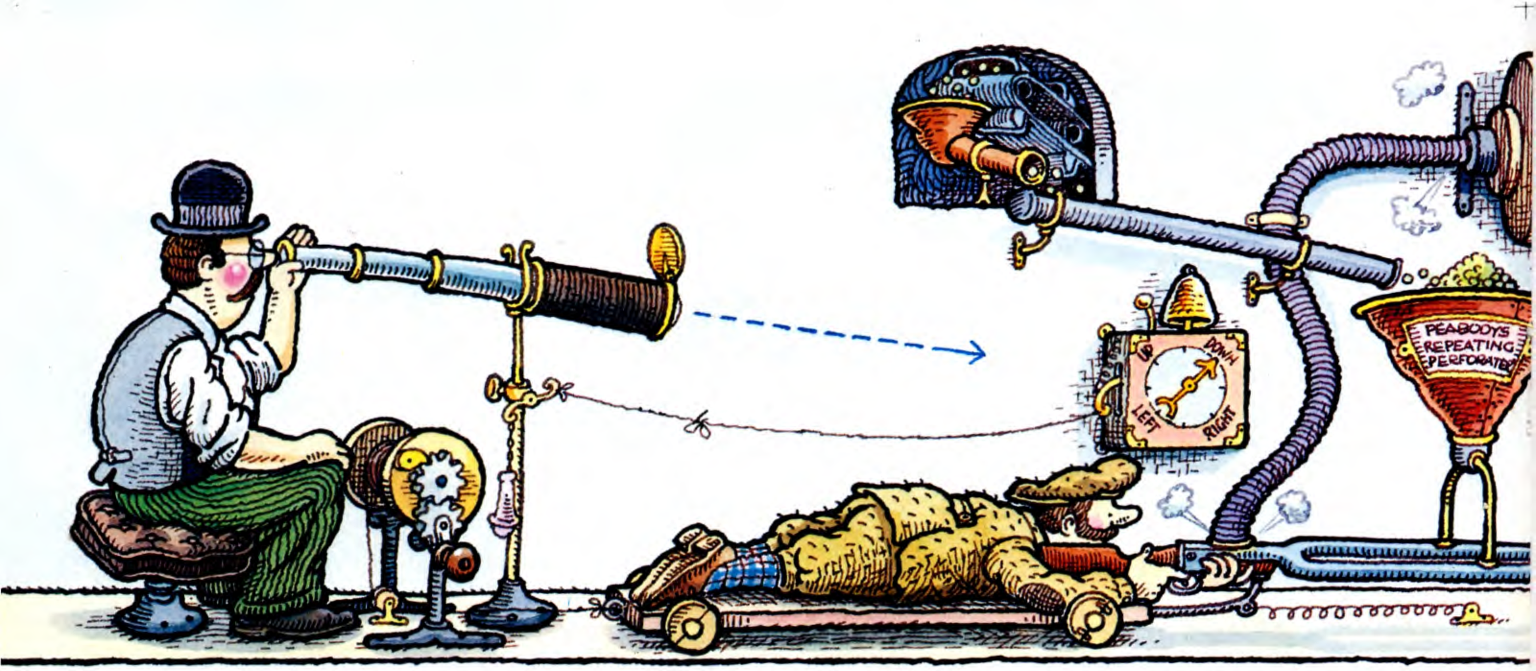
Once you have toyed with this program for a while, and seen how lengthening the lever lessens the effort, you should be able to recognize levers at work in any machine. This lessening of the effort is properly called mechanical advantage, and is given by dividing the effort into the load. If you rearrange the formula above into this form, you can show that the mechanical advantage is equal to the distance moved by the effort divided by distance moved by the load. So next time you pull a lever—a door handle or bicycle brake—notice that the travel of the effort end is far greater than at the other end.



REACHING GREATER HEIGHTS

Pushing and pulling are best done by levers, but when you need to raise a load to great height, the lever must be modified. A system of pulleys is just such a device—it gives mechanical advantage. Key and RUN the next program to see this demonstrated:

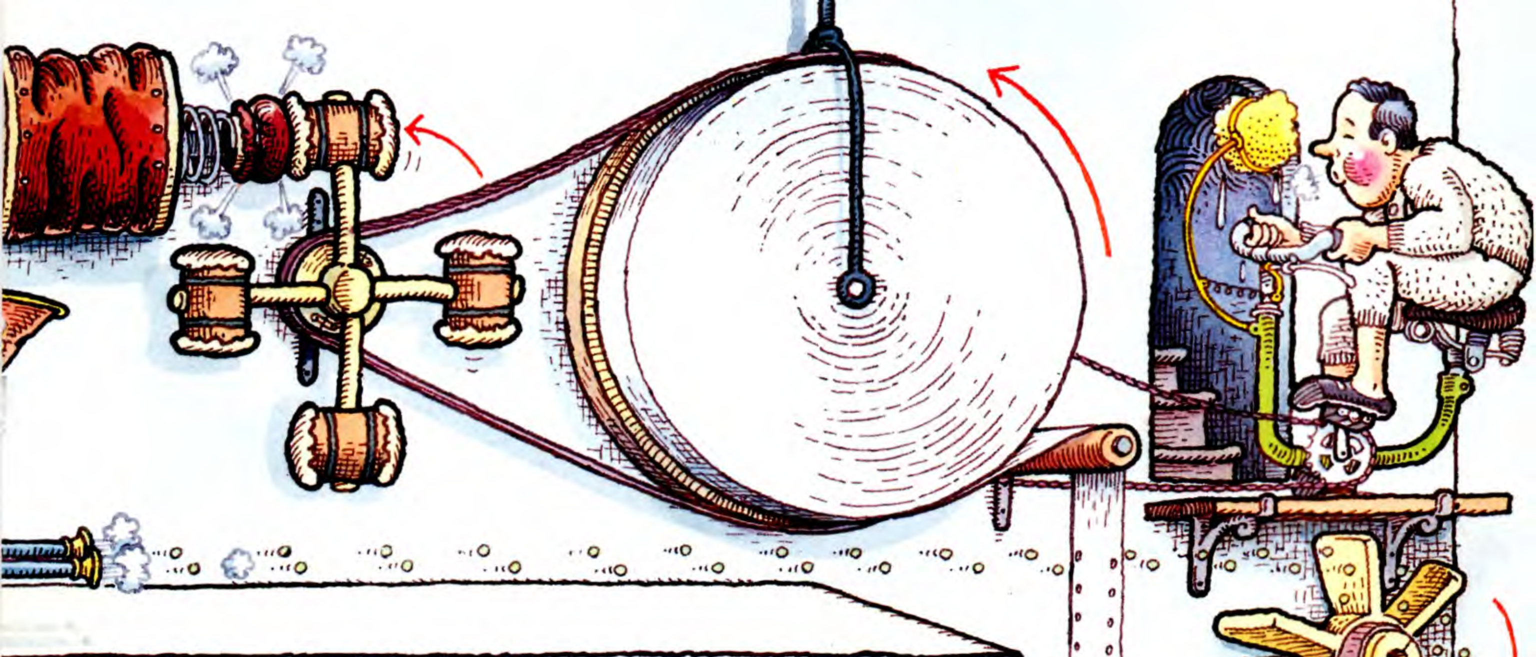
```
10 CLEAR 32399: RESTORE : GOSUB 51Ø:
   BORDER Ø: PAPER Ø: INK 7: CLS
20 PRINT AT 1Ø,Ø;"How many pulleys (2,4 OR
   6) ?"
30 LET a$=INKEY$: IF a$<>"2" AND
   a$<>"4" AND a$<>"6" THEN GOTO
   3Ø
40 PRINT TAB (1Ø);a$: LET np=VAL (a$):
   LET l=25*np-5Ø
```




```
45 IF np=2 THEN POKE 32431,25
46 IF np=4 THEN POKE 32431,17
47 IF np=6 THEN POKE 32431,14
50 PRINT : PRINT "Effort required=";INT
   (1000/np): PRINT "Kilograms to lift 1
   Tonne"
55 FOR m=1 TO 500: NEXT m
60 CLS : GOSUB 1000
70 LET sp=120
90 FOR k=1 TO 50
100 RANDOMIZE USR 32400
105 PLOT OVER 1;191-l-np,sp: DRAW
   OVER 1;8,0
110 PLOT OVER 1;191-l-np,sp: DRAW
   OVER 1;8,0
120 LET sp=sp-np: IF sp<=0 THEN LET
   sp=120
150 NEXT k
160 IF INKEY$="" THEN GOTO 160
170 RUN
510 FOR n=32400 TO 32491
520 READ a: POKE n,a
530 NEXT n
540 DATA 62,60,79,230,192,15,15,15,
   198,64,103,121,230,7,132,103,
   121,135,135,230,224,111,62
550 DATA 175,254,192,208,145,216,8,
   14,14,125,177,111,62,30,254,32,208,
   145,216,60,79,6,0,197,229,17,224,91,
   237,176,225
560 DATA 193,217,8,167,40,30,71,217,
   124,60,87,93,230,7,32,10,123,198,32,
   95,56,4,122,214,8,87,235
570 DATA 229,197,237,176,193,225,
   217,16,227,217,201
580 RETURN
1000 PLOT 0,170: DRAW 255,0
1010 FOR k=1 TO np STEP 2
1020 CIRCLE (232-k*26),140,13
1030 CIRCLE (258-k*26),50,13
1040 NEXT k
1050 PLOT 245,170: DRAW 0,-125
1060 FOR k=1 TO np-1
1070 PLOT 246-k*26,140: DRAW 0,-90
1080 NEXT k
1090 PLOT 197-l-np,140: DRAW 0,-140
1100 PLOT 208,141: DRAW (256-
   np*26)-PEEK 23677,0
1110 PLOT 234,50: DRAW (282-
   np*26)-PEEK 23677,0
1120 PLOT 206,170: DRAW 0,-28: PLOT
   258-np*26,170: DRAW 0,-28
1130 PLOT 231-l*2/3,50: DRAW 0,-20
1140 PLOT 232-l/3,50: DRAW 0,-20
1145 DRAW (231-l*2/3)-PEEK 23677,0
1150 PLOT 231-l/2,29: DRAW 0,-10
1160 DRAW -9,0: DRAW 0,-10: DRAW
   19,0: DRAW 0,10: DRAW -9,0
1170 PLOT 180-l,0: DRAW
   (180-l-20/SQR (np))-PEEK 23677,0:
   DRAW 0,(20/SQR (np))-PEEK 23678:
   DRAW (180-l)-PEEK 23677,0: DRAW
   0,0-PEEK 23678
1210 RETURN
```

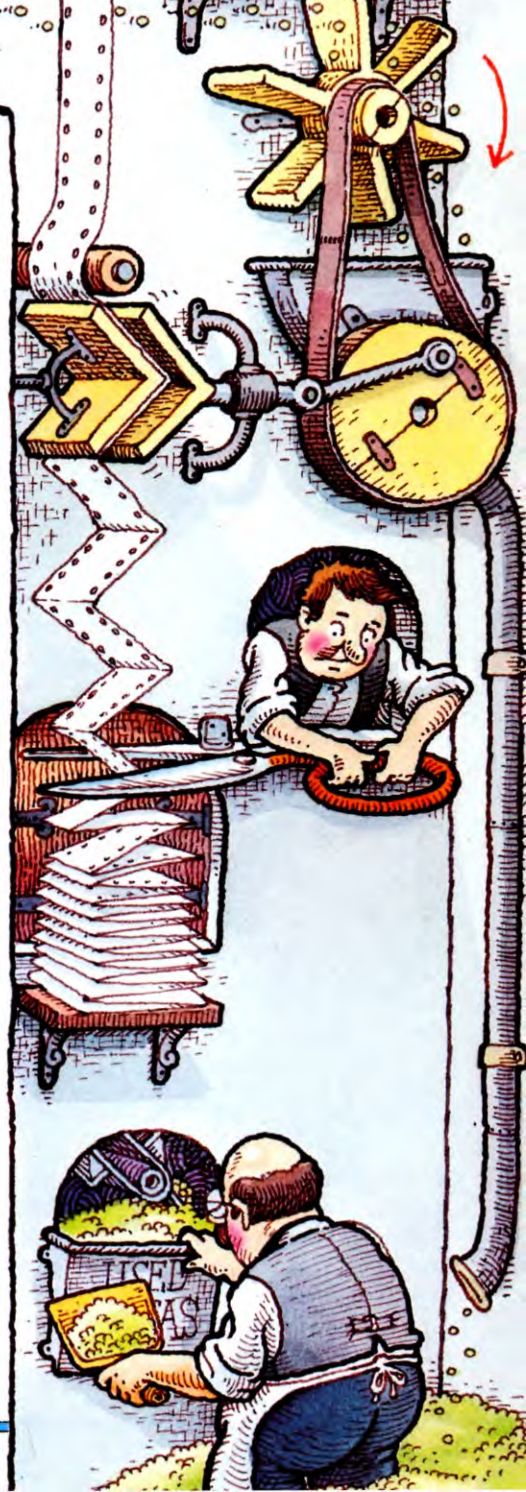
```
20 A$="▨▟□▄▧▟▌▌▌▌▌▟
   ▙'▙▄▟▌▌▌▌▌▧▟□▄▨"
30 FOR Z=1 TO 20:D$=D$+"▟":
   NEXT Z
40 B$=A$+"○○▙▙▙▙▙"
   +A$+"○○▙▙▙▙▙▙"+A$
50 C$="▩▟□□▄▙▟□□□
   □□□□▄▙▟□□□□□□□
   □▄▙▟□□□▄"
60 POKE 53280,7:POKE 53281,7:
   GOSUB 1000
90 S=0:FORZ=18TO6STEP-.5:PRINT
   "▤◧"LEFT$(D$,Z)TAB(19)
   B$:PRINT"○○"SPC(18)C$
100 PRINTTAB(18)"▟◳▙▙▙▙◰
   □◫□◳▙▙▙◰□◫□
   ◳▙▙▙◰"
110 PRINTTAB(18)"□□□□□□□◳▤
   ■▟□◲□◱□▄▟◰□□□
   □□□"
120 PRINTTAB(24)"□□■▟□
   ⊞□⊞□▄□□"
130 PRINTTAB(24)"□□□□□
   □□□□":FOR D=1 TO 200:
   NEXT D
140 POKE 1278+40*S,93:S=S+3:
   IF S>15 THEN S=0
150 POKE 1278+40*S,220:NEXT Z:END
1000 PRINT"◘■";:FOR Z=1 TO
   40:PRINT "▟□▄";:NEXT
1003 FOR Z=1 TO 3:PRINTSPC(20)
   "▟◫▙▙▙▙▙▙▙▙
   ◫▙▙▙▙▙▙▙▙▙▙
   ◫":NEXT Z
1005 PRINT"▤▟▟▟"SPC(14)"
   ◲▙▙▙◱▙▙▙▙◲▙▙▙
   ◱▙▙▙◲▙▙▙▙◱▙▙▙
   ◫"
1006 FOR Z=1 TO 16:PRINTSPC(14)
   "◫▙▙▙◫▙▙▙▙◫▙▙▙
   ◫▙▙▙◫▙▙▙▙◫▙▙▙
   ◫":NEXT Z
1007 PRINTSPC(14)"◫▟▌▌
   ◫▟▌◫"
1008 PRINT"○○"TAB(10)"▟
   ▟■□▄□"
1010 PRINT"▤◧▟▟▟"TAB(15)A$
   "▤▟▟▟"TAB(23)A$"▤▟
   ▟▟"TAB(31)A$
1011 PRINT"▤▟▟▟▟"SPC
   (14)C$
1020 PRINT"▤◨▟EFFORT REQUIRED
   =":PRINT"166.6 KILOGRAMS
   TO":PRINT"LIFT 1 TONNE."
1030 RETURN
```

```
5 MODE1:VDU19,3,2,0,0,0
10 VDU23,224,1,15,31,63,63,127,127,
   254,23,225,254,127,127,63,63,31,15,
   1,23,226,127,254,254,252,252,248,224,
   128,23,227,128,224,248,252,252,254,
   254,127
```



```
15 A$ = CHR$(224) + CHR$(227) + CHR$
   (8) + CHR$(8) + CHR3(1Ø) + CHR$(225)
   + CHR$(226)
2Ø PRINT'"HOW MANY PULLEYS (2,4, OR 6)
   ?";
3Ø G$ = GET$:IF G$ < > "2" AND
   G$ < > "4" AND G$ < > "6" THEN 3Ø
4Ø PRINTG$:NP = VAL(G$):L = 125*
   NP - 5Ø
5Ø PRINT'"EFFORT REQUIRED
   = ";1ØØØ/NP'"KILOGRAMS TO LIFT 1
   TONNE":K = INKEY(1ØØØ)
6Ø CLS:PROCSETUP
7Ø SP = 74Ø
9Ø FORL = 3ØØTO7ØØ STEP4:FORK = 1
   TONP STEP 2
11Ø GCOLØ,2:MOVE 86Ø - NP*6Ø,77Ø:
   DRAW 86Ø - NP*6Ø,Ø
12Ø SP = SP - 4*NP:IF SP < Ø THEN SP = 74Ø
13Ø MOVE 855 - NP*6Ø,SP:DRAW 875
   - NP*6Ø,SP:GCOLØ,1:PROCWEIGHT
14Ø GCOLØ,Ø:MOVE 862 - K*6Ø,L - 4:
   PRINTA$:GCOLØ,3:MOVE 862 -
   K*6Ø,L:PRINT A$
142 GCOLØ,Ø:PROCWEIGHT
145 MOVE 855 - NP*6Ø,SP:DRAW 875 -
   NP*6Ø,SP
15Ø NEXTK,L
16Ø GCOLØ,1:PROCWEIGHT:GCOLØ,2:
   MOVE 86Ø - NP*6Ø,77Ø:DRAW 86Ø -
   NP*6Ø,Ø:MOVE 855 - NP*6Ø,SP:
   DRAW 875 - NP*6Ø.SP:VDU 4: VDU 2Ø
17Ø G = GET:RUN
1ØØØ DEF PROCSETUP
1Ø1Ø VDU24,2ØØ;94Ø;1Ø8Ø;96Ø;:
   GCOLØ,13Ø:CLG:VDU26:VDU5
1Ø2Ø FOR K = 1 TO NP STEP 2
1Ø3Ø MOVE (8ØØ - K*6Ø),8ØØ:PRINT A$
1Ø4Ø MOVE (862 - K*6Ø),3ØØ:PRINT A$
1Ø5Ø NEXT:GCOLØ,2
1Ø6Ø MOVE 86Ø,937:DRAW 86Ø,28Ø
1Ø7Ø FOR K = 1 TO NP - 1
1Ø8Ø MOVE (86Ø - K*6Ø),275:DRAW
   (86Ø - K*6Ø),77Ø
1Ø9Ø NEXT
11ØØ MOVE 86Ø - NP*6Ø,77Ø:DRAW
   86Ø - NP*6Ø,Ø
111Ø GCOLØ,1:MOVE 78Ø,776:MOVE
   78Ø,766:PLOT 85,885 - NP*6Ø,776:PLOT
   81,Ø, - 1Ø
112Ø L = 3ØØ:PROCWEIGHT
113Ø MOVE 772,768:DRAW772,937:
   MOVE 892 - NP*6Ø,768:DRAW 892 -
   NP*6Ø,937
114Ø MOVE 84Ø - NP*6Ø,Ø:MOVE 84Ø -
   NP*6Ø,6Ø/SQR(NP):PLOT81, - 6Ø/
   SQR(NP), - 6Ø/SQR(NP):PLOT81,Ø,
   6Ø/SQR(NP)
115Ø ENDPROC
116Ø DEF PROCWEIGHT
117Ø MOVE834,L - 32:DRAW834,L - 15Ø:
   DRAW 951 - NP*6Ø,L - 15Ø:DRAW951 -
   NP*6Ø,L - 32
118Ø MOVE893 - NP*3Ø,L - 15Ø:DRAW 893
   - NP*3Ø,L - 18Ø:PLOTØ, - 4Ø,Ø
119Ø PLOTØ,Ø - 4Ø:PLOT81,8Ø,4Ø:
   PLOT81,Ø, - 4Ø
12ØØ IF NP = 6 THEN MOVE 712,L - 32:
   DRAW 712,L - 15Ø
121Ø ENDPROC
```

```
1Ø PMODE3,1:DIMP(28Ø)
2Ø CLS:PRINT"HOW MANY PULLEYS (2,4 OR
   6) ?";
3Ø A$ = INKEY$:IF A$ < > "2" AND
   A$ < > "4" AND A$ < > "6" THEN3Ø
4Ø PRINTA$:NP = VAL(A$):L = 25*NP
   - 5Ø
5Ø PRINT:PRINT"EFFORT REQUIRED
   = ";1ØØØ/NP:PRINT"KILOGRAMS
   TO LIFT 1 TONNE":FORG = 1TO
   4ØØØ:NEXT
6Ø PCLS:SCREEN1,Ø:GOSUB1ØØØ
7Ø GET(249,172) - (215 - L,1Ø6),P,
   G:SP = 38
```



```
80 COLOR4,1
90 FORK=106TO49 STEP-1
100 PUT(249,K+66)-(215-L,K),
   P,PSET
110 LINE(191-L-NP,SP)-(199-
   L-NP,SP),PRESET
120 SP=SP+NP:IF SP>191 THEN SP=38
130 LINE(195-L-NP,33)-(195-
   L-NP,191),PSET
140 LINE(191-L-NP,SP)-(199-
   L-NP,SP),PSET
150 NEXT
160 IF INKEY$="" THEN 160
170 RUN
1000 LINE(0,0)-(255,10),PSET,BF
1010 FORK=1TONP STEP2
1020 CIRCLE(232-K*26,33),13,
   2:PAINT(232-K*26,33),2
1030 CIRCLE(232-K*26,33),14,
   4,1,.5,1
1040 CIRCLE(258-K*26,121),13,
   2:PAINT(258-K*26,121),2
1050 CIRCLE(258-K*26,121),14,
   4,1,0,.5
1060 NEXT
1070 LINE(245,10)-(245,121),PSET
1080 FORK=1TONP-1
1090 LINE(246-K*26,33)-(246-
   K*26,121),PSET
1100 NEXT
1110 LINE(195-L-NP,33)-(195
   -L-NP,191),PSET
1120 COLOR4,3:LINE(208,32)-
   (256-NP*26,34),PRESET,BF
1130 LINE(234,120)-(282-NP*26,
   122),PRESET,BF
1140 LINE(206,10)-(206,33),
   PSET:LINE(258-NP*26,10)-(258
   -NP*26,33),PSET
1150 LINE(232-L*2/3,121)-
   (232-L*2/3,141),PRESET
1160 LINE(232-L/3,121)-(232-
   L/3,141),PRESET
1170 LINE-(232-L*2/3,141),
   PRESET
1180 LINE(232-L/2,141)-(232-
   L/2,151),PRESET
1190 DRAW"L9D19R19U19L9"
1200 LINE(180-L,191)-(180-L
   -20/SQR(NP),191-20/SQR(NP)),
   PRESET,BF
1210 RETURN
```

The program gives a prompt (Line 20) for you to specify the number of pulleys you wish to see demonstrated. On the Commodore 64, the program is kept short by having only one demonstration of six pulleys, so the choice is omitted for these micros.

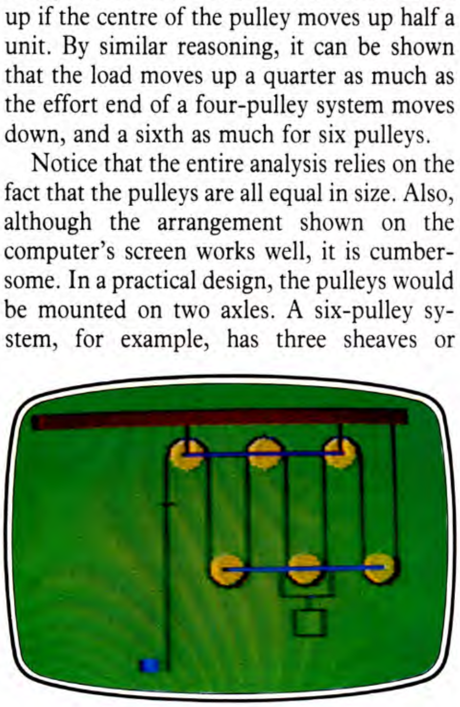
Enter 2 to begin with, and the effort to raise a load of 1000 kg (1 tonne) will be printed (Line 50) on the screen. Next the program animates the load being raised. The routine between Lines 1000 and 1210 (called at Line 60) draws the system of ropes, fixed pulley and supports, and Lines 90 to 150 draw the movable pulley, load and rope in motion.

Now RUN the program again, but enter 4, then 6, in response to the prompt, and compare the value for effort and the distance moved by the free end of the rope. Since the effort equals the load divided by the number of pulleys, the effort will be 1/2, 1/4 or 1/6 of 1000 kg for 2, 4 or 6 pulleys. So the greater the number of pulleys, the less the effort required to raise the same load. As with the lever, the mechanical advantage is given by load/effort. For two pulleys, this is 1000 divided by 500, which equals 2. Similarly, the mechanical advantage is 4 for four pulleys and 6 for six pulleys.

SAVING EFFORT

As for the lever, the relationship between the distance moved by load and effort also holds good. Although, in the case of two pulleys, say, the animation shows that the effort end of the rope moves down twice the distance moved up by the load, it is not obvious how this can be verified. If you consider that the rope on the lefthand side of the single, fixed pulley moves down one unit, then the rope on the righthand side also has to move up two units. The difference with the single, fixed pulley is that the rope on its righthand side is fixed, so a travel of one unit on the left is taken up if the centre of the pulley moves up half a unit. By similar reasoning, it can be shown that the load moves up a quarter as much as the effort end of a four-pulley system moves down, and a sixth as much for six pulleys.

Notice that the entire analysis relies on the fact that the pulleys are all equal in size. Also, although the arrangement shown on the computer's screen works well, it is cumbersome. In a practical design, the pulleys would be mounted on two axles. A six-pulley system, for example, has three sheaves or grooved wheels on an axle within a support frame, which can be secured to a crane or rigid overhead structure. A rope attached to the bottom of the frame is threaded through a similar system of sheaves, and the load is attached to the bottom of its frame. This is the arrangement of the conventional block and tackle used on all cranes.



Pulleys simulated on Dragon

THE HYDRAULIC RAM

An equally important application of mechanical advantage is in the hydraulic ram or jack. This is the device used to move certain parts on all sort of machinery—from garbage compactor doors and tipper lorry bodies to aircraft landing gear and even car brakes. The car hydraulic braking system illustrates well how a small effort but long travel—at the pedal—can be magnified to a force strong enough to stop the wheels even at high speed. Enter and RUN the next program to see the principle demonstrated:

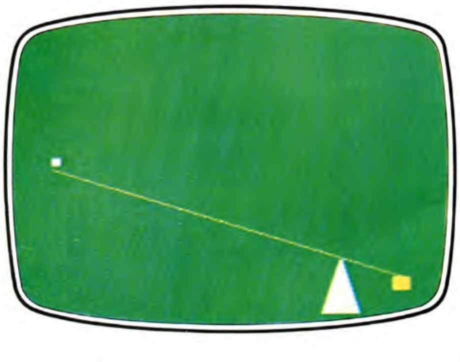
```
30 BORDER 0: PAPER 7: INK 0: CLS
50 GOSUB 300
90 INPUT "Plunger travel (1-90) ?□□";tr
100 IF tr<1 OR tr>90 THEN GOTO 90
110 FOR k=1 TO tr
120 PLOT 40,128-(k-1): DRAW INK
   7;10,0: PLOT 40,128-k: DRAW 15,0
130 PLOT 175,127+(k-1)/10: DRAW INK
   1;56,0: PLOT 175,127+k/10: DRAW INK
   1;56,0
135 PRINT INK 0;AT 3,5;k; INK 0;AT 3,25;INT
   (k/10)
140 NEXT k
150 INK 0: PRINT AT 21,0;"□□
   □□□□□□Again ? (y OR n)
   □□□□□"
160 IF INKEY$="y" THEN RUN
170 IF INKEY$<>"n" THEN GOTO 160
180 STOP
300 FOR n=6 TO 18: PRINT PAPER 1;AT
   n,5;"□□": NEXT n
310 FOR n=6 TO 18: PRINT PAPER 1;
   AT n,22;"□□□□□□□": NEXT n
320 FOR n=18 TO 21: PRINT PAPER
   1;AT n,5;"□□□□□□□□□□
   □□□□□□□□□□□□
   □": NEXT n
330 PLOT 39,155: DRAW 0,-155: DRAW
   192,0: DRAW 0,155: PLOT 56,155: DRAW
   0,-124: DRAW 120,0: DRAW 0,124
340 PLOT 40,127: DRAW -2,0: PRINT AT
   6,3;"0": PLOT 40,37: DRAW -2,0: PRINT
   AT 17,2;"90": PLOT 232,127: DRAW 2,0:
   PRINT AT 6,30;"0"
350 PLOT 232,137: DRAW 2,0: PRINT AT
   4,30;"9"
360 FOR n=127 TO 37 STEP -10
```

```
370 PLOT 40,n: DRAW −2,Ø: NEXT n
380 PLOT 232,132: DRAW 2,Ø
390 PLOT 12Ø,35: DRAW Ø,−35
400 PLOT 11Ø,4Ø: DRAW 2Ø,Ø: DRAW −5,5:
   DRAW 5,−5: DRAW −5,−5
41Ø INK 7
44Ø RETURN
```

```
1Ø HIRES Ø,1
2Ø MULTI Ø,5,6
3Ø COLOUR 2,7
4Ø DIM P(4),R(31)
5Ø GOSUB 3ØØ
8Ø GET A$:IF A$="" THEN 8Ø
9Ø CSET(Ø):INPUT "♥PLEASE GIVE
   PLUNGER TRAVEL (1-9Ø)";TR
1ØØ IF TR<1 OR TR>9Ø THEN 9Ø
11Ø CSET(2):MULTI Ø,5,6:FOR K=1 TO TR
115 LINE 16,4Ø+K,2Ø,4Ø+K,Ø
12Ø BLOCK 16,41+K,2Ø,48+K,2
125 LINE 91,53−K/7,111,53−K/7,3
13Ø BLOCK 91,23−K/7,111,48−K/7,2
14Ø NEXT K
15Ø GET A$:IF A$="" THEN 15Ø
16Ø CSET(Ø):PRINT "♥AGAIN (Y/N) ?"
17Ø GET A$:IF A$<>"Y" AND
   A$<>"N" THEN 17Ø
18Ø IF A$="Y" THEN RUN
19Ø PRINT "♥":END
3ØØ LINE 15,4Ø,15,2ØØ,1
3Ø5 LINE 112,4Ø,112,2ØØ,1
31Ø BLOCK 21,4Ø,9Ø,18Ø,1
33Ø TEXT 5Ø,17Ø,">",Ø,1,8
34Ø BLOCK 16,41,2Ø,48,2
35Ø BLOCK 91,23,111,48,2
355 PAINT 8Ø,19Ø,3:LINE 55,
   18Ø,55,2ØØ,Ø
36Ø FOR K=Ø TO 9
37Ø LINE 13,49+K*1Ø,15,49+K*1Ø,1
38Ø NEXT K
39Ø TEXT 5,46,"Ø",1,1,8
395 TEXT Ø,14Ø,"9Ø",1,1,8
4ØØ FOR K=Ø TO 9 STEP 3
41Ø LINE 113,49−K,115,49−K,1
42Ø NEXT K
43Ø TEXT 116,48,"Ø",1,1,8
435 TEXT 116,35,"9",1,1,8
44Ø RETURN
```

```
1Ø MODE 1
2Ø VDU19,Ø,4;Ø;
3Ø REPEAT:GCOLØ,2
5Ø PROCRam:PROCFluid
6Ø REPEAT:INPUT"ENTER TRAVEL OF
   PLUNGER (Ø-44Ø)□"D:UNTIL D>=Ø
   AND D<=44Ø
7Ø GCOLØ,1:FORK=1TOD:MOVE2Ø5,
   8ØØ−K:PLOT3,93,Ø
8Ø MOVE7Ø5,8ØØ+K/1Ø:PLOT1,3Ø9,
   Ø:NEXT
9Ø INPUT"Again?(Y/N)"K$
1ØØ IF K$="Y" THEN CLG ELSE IF
   K$="N" THEN END ELSE 9Ø
11Ø UNTIL FALSE
12Ø END
13Ø DEF PROCColumn(X,Y,XS,YS,C)
14Ø VDU24,X;Y;XS;3ØØ+YS;
15Ø GCOLØ,128+C:CLG
16Ø VDU24,X;3Ø3+YS;XS;9ØØ;
17Ø GCOLØ,128:CLG
18Ø VDU26:ENDPROC
19Ø DEF PROCRectangle(X,Y,XS,YS,C)
2ØØ VDU24,X;Y;XS;YS;
21Ø GCOLØ,128+C:CLG
22Ø VDU26:ENDPROC
23Ø DEF PROCRam
24Ø MOVE 2ØØ,9ØØ:DRAW 2ØØ,3ØØ:
   DRAW 1Ø16,3ØØ:DRAW 1Ø16,9ØØ
25Ø MOVE 3ØØ,9ØØ:DRAW 3ØØ,35Ø
26Ø DRAW 7ØØ,35Ø:DRAW 7ØØ,9ØØ
27Ø VDU5:FOR Y=41Ø TO 81Ø STEP
   1ØØ:MOVE 18Ø,Y:PRINT"−":NEXT
28Ø MOVE 1Ø16,8ØØ:DRAW 1Ø36,8ØØ:
MOVE 1Ø16,84Ø:DRAW 1Ø36,84Ø
29Ø MOVE 14Ø,81Ø:PRINT"Ø":MOVE
   8Ø,41Ø:PRINT"4ØØ"
3ØØ MOVE 1Ø5Ø,81Ø:PRINT"Ø":MOVE
   1Ø5Ø,85Ø:PRINT"4Ø"
31Ø MOVE 13Ø,97Ø:PRINT"PLUNGER":
   MOVE 8ØØ,97Ø:PRINT"RAM"
32Ø MOVE 45Ø,375:DRAW 55Ø,275:
   MOVE 45Ø,275:DRAW 55Ø,375
33Ø VDU4:ENDPROC
34Ø DEF PROCFluid
35Ø PROCColumn(2Ø5,3Ø5,298,5ØØ,1)
36Ø PROCRectangle(298,3Ø5,7ØØ,
   345,1)
37Ø PROCColumn(7Ø5,3Ø5,1Ø14,
   5ØØ,1)
38Ø VDU5:MOVE 2Ø5,9ØØ:PRINT"1kg"
39Ø MOVE 8ØØ,9ØØ:PRINT"1Ø kg"
4ØØ VDU4:VDU28,Ø,31,39,31:
   ENDPROC
```

```
1Ø PMODE3,1
```



The lever display on Acorn

```
2Ø PCLS
3Ø SCREEN1,Ø
4Ø DIM P(4),R(31)
5Ø GOSUB3ØØ
6Ø GET(32,52)−(43,38),P,G
7Ø GET(182,52)−(223,23),R,G
8Ø IF INKEY$="" THEN8Ø
9Ø CLS:INPUT"PLEASE GIVE PLUNGER
   TRAVEL□□□□□□(1-9Ø)";TR
1ØØ IF TR<1 OR TR>9Ø THEN9Ø
11Ø SCREEN1,Ø:FORK=1TOTR
12Ø PUT(32,52+K)−(43,38+K),
   P,PSET
13Ø PUT(182,52−K/1Ø)−(223,23−
   K/1Ø),R,PSET
14Ø NEXT
15Ø IF INKEY$="" THEN 15Ø
16Ø CLS:PRINT"AGAIN (Y/N) ?"
17Ø A$=INKEY$:IF A$<>"Y" AND
   A$<>"N" THEN 17Ø
18Ø IF A$="Y" THENRUN
19Ø CLS:END
3ØØ DRAW"BM3Ø,3ØC2D131R195U
   131BL45D2ØNR45D96L135U96NL
   15U2Ø"
31Ø PAINT(2ØØ,7Ø),3,2
32Ø DRAW"BM32,5ØC3R1ØBR14ØR41"
33Ø DRAW"BM127,161C2U2ØBH8C4R
   16NH5G5"
34Ø LINE(34,48)−(41,41),PSET,BF
35Ø LINE(191,48)−(215,23),PSET,
   BF
36Ø FORK=ØTO9
37Ø COLOR2:LINE(26,5Ø+K*1Ø)
   −(3Ø,5Ø+K*1Ø),PSET
38Ø NEXT
39Ø DRAW"BM2Ø,46C4D6L4U6R4BD
   9ØNL4D6L4U6BL4ND6L4D3R4C2"
4ØØ FORK=ØTO9 STEP 3
41Ø LINE(225,5Ø−K)−(229,5Ø−K),
   PSET
42Ø NEXT
43Ø DRAW"BM232,48C4D6R4U6L4
   BU9NR4U3R4D6"
44Ø RETURN
```

When you RUN this program, Line 5Ø branches it to a routine (two for the Acorns) to draw the ram. You are then asked to specify the travel of the plunger—equivalent to the pedal on a braking system. The relative movement of liquid in the ram is then animated.

The travel is a measure of the effort applied to the plunger, but the mechanical advantage of the system is determined by the diameter of the plunger and the load piston. If both are of the same diameter, there is no advantage, but as the diameter of the plunger is reduced, so the ease of raising a load on the righthand side of the system increases.

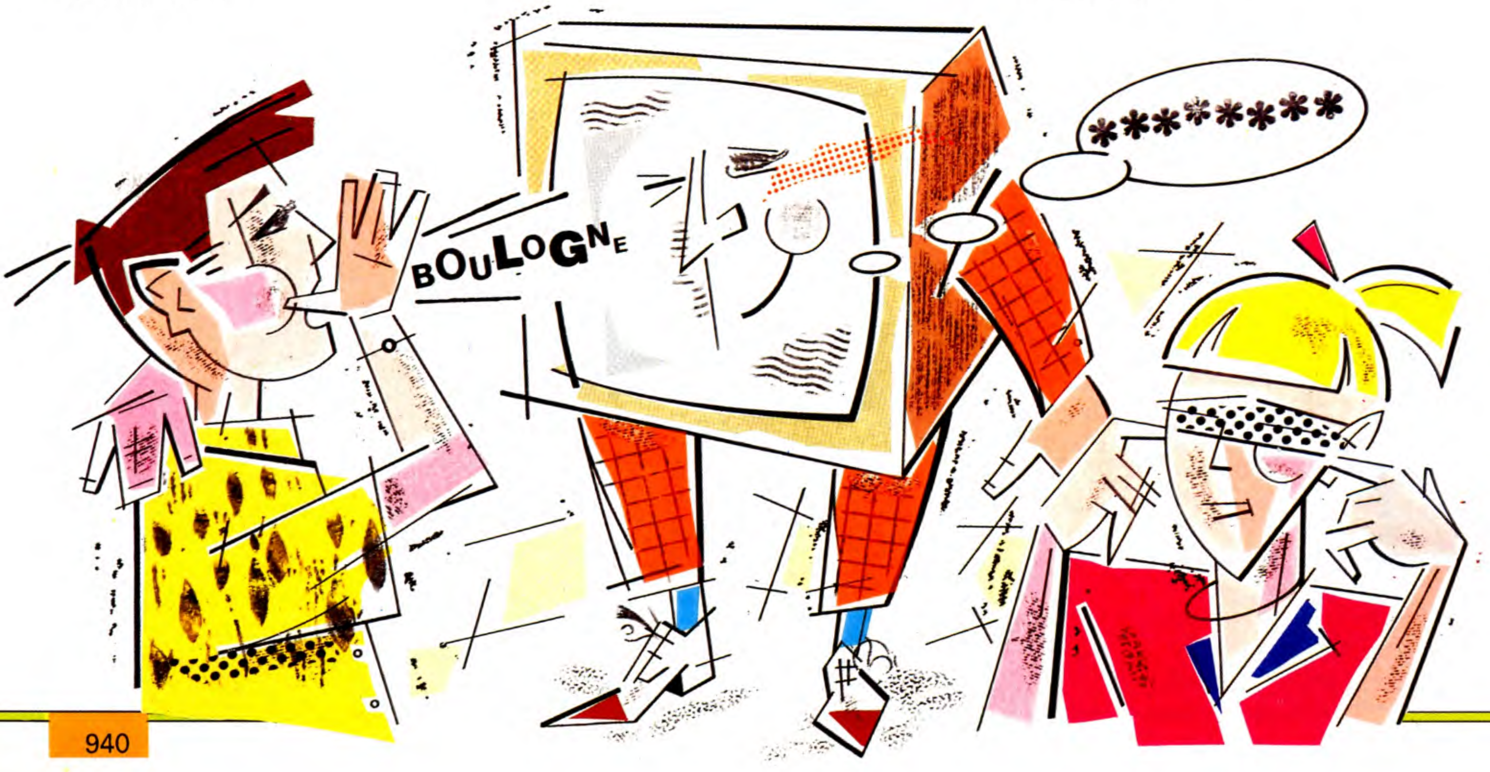
GETTING IT WORD-PERFECT

Resurrect zinziberaceous, qintar, xylem and qinquagesima from the dusty corners of your dictionary. Use dirty tactics to gain the edge over your friends in the word game

In this second and final article about *INPUT*'s word game there is all you need to start playing. Type the remaining lines in to see some interesting and fun applications for your machine's string handling. Then try to baffle your friends with obscure words and phrases.

There are routines for each of the game's three options—buying letters, guessing a specific letter at a specific position, and guessing the complete phrase. The program also keeps track of the score, number of guesses, and the number of turns.

```
370 IF d$<>"XX" AND d$<>"ZZ"
    AND LEN d$>1 THEN GOTO 36Ø
38Ø IF d$=CHR$ 32 THEN GOTO 41Ø
385 IF d$="ZZ" THEN LET d$="": GOTO
    9ØØ
39Ø IF CODE d$<65 OR CODE d$>9Ø
    THEN GOTO 36Ø
4ØØ IF d$="XX" THEN LET d$="": GOTO
    5ØØ
41Ø GOSUB 79Ø
42Ø LET e=Ø
43Ø LET e=e+1
44Ø IF e=l+1 THEN LET tb=tb-g: LET
    q$(m TO m+7)="": PRINT AT 4,Ø;q$:
    LET d$="": GOTO 47Ø
45Ø IF s$(e)<>d$ THEN GOTO 43Ø
46Ø IF s$(e)=d$ THEN LET z$(e)=d$:
    GOTO 43Ø
47Ø PAUSE 1ØØ: PRINT AT 14,Ø;:
    FOR r=1 TO 7: PRINT "□□□□
    □□□□□□□□□□□□□□": 
    NEXT r
48Ø PRINT PAPER 2; INK 6;AT 1,22;tb;CHR$
    32: PRINT PAPER 2; INK 6;AT 14,Ø;z$:
    PRINT "GUESS□";f: LET f=f+1: IF
    s$=z$ THEN GOTO 73Ø
49Ø GOTO 36Ø
5ØØ INPUT "WHAT CHARACTER DO YOU
    WANT TO□□GUESS?", LINE d$
51Ø IF LEN d$>1 THEN GOTO 5ØØ
52Ø IF d$=CHR$ 32 THEN GOSUB 79Ø:
    GOTO 55Ø
53Ø IF CODE d$<65 OR CODE d$>9Ø
    THEN GOTO 5ØØ
54Ø GOSUB 79Ø
55Ø PRINT PAPER 2; INK 6;AT 18,Ø;d$: PRINT
    AT 18,2;"AT WHAT POSITION? USE L/R
    CURSOR KEYS AND PRESS Ø TO ENTER"
56Ø PRINT PAPER 2; INK 6; AT 14,Ø;z$:
    PRINT PAPER 6; INK 2; AT 14,b;z$(b+1)
57Ø PAUSE Ø: LET y$=INKEY$: IF y$=""
    THEN GOTO 57Ø
59Ø IF y$="8" AND b<l-1 THEN LET
    b=b+1
6ØØ IF y$="5" AND b>Ø THEN LET
    b=b-1
61Ø IF y$="Ø" THEN GOTO 68Ø
64Ø IF b>=32 THEN LET w=15: LET
    v=b-32
65Ø IF b<32 THEN LET w=14: LET v=b
66Ø PRINT PAPER 2; INK 6;AT 14,Ø;z$: PRINT
    PAPER 6; INK 2;AT w,v;z$(b+1)
67Ø GOTO 57Ø
68Ø IF z$(b+1)<>"*" THEN GOTO 57Ø
69Ø IF s$(b+1)<>d$ THEN LET
    tb=tb-g/2: PRINT FLASH 1;AT
    17,Ø;"BAD LUCK": PAUSE 5Ø: LET b=Ø
    GOTO 47Ø
7ØØ IF s$(b+1)=d$ THEN PRINT FLASH
    1;AT 17,Ø;"GOOD GUESS": PAUSE 5Ø:
    LET z$(b+1)=d$: LET tb=tb+g: LET
    b=Ø
71Ø IF s$=z$ THEN GOTO 73Ø
72Ø GOTO 47Ø
```

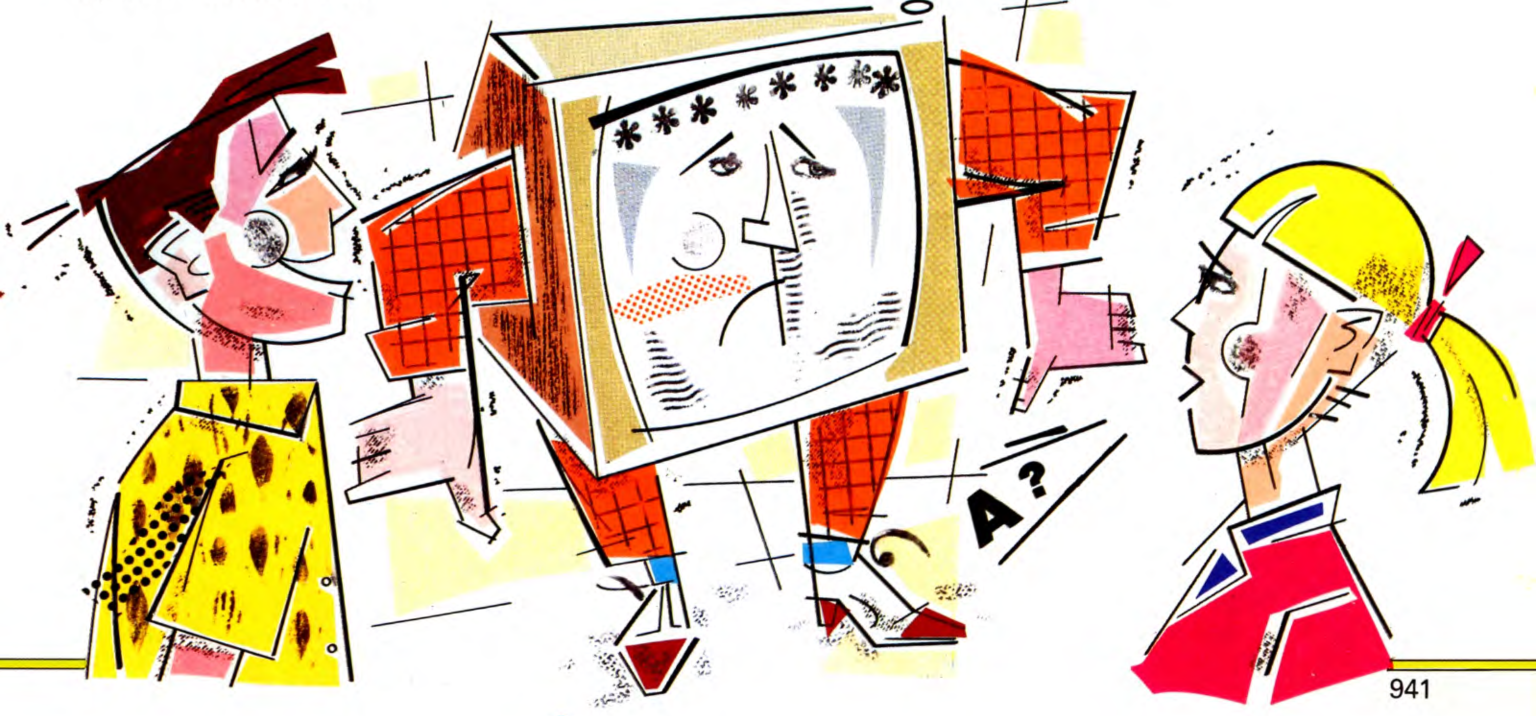


- COMPLETE *INPUT'S* WORD GAME WITH NEW ROUTINES
- BUYING LETTERS
- CHECKING THE GUESSER'S INPUT
- PUTTING A SPECIFIC LETTER IN POSITION
- GUESSING THE COMPLETE PHRASE
- SKULDUGGERY!

```
730 PRINT INK 6; PAPER 2;AT 1,22;tb: PRINT
    AT 17,0;"CONGRATULATIONS,□";b$;TAB
    0;TAB 31;"□": PAUSE 100: CLS
740 LET k=k+1: IF k=t*2 THEN GOTO 880
750 LET c$=a$: LET a$=b$: LET b$=c$
760 LET tc=ta: LET ta=tb: LET tb=tc
770 LET q$="": LET d=0: LET f=1
780 GOTO 160
790 LET m=(CODE d$-64)*8-7
800 IF m=-263 THEN LET m=209
810 IF q$(m TO m+5)=" " THEN GOTO
    360
820 LET g=VAL q$(m+2 TO m+3)
830 RETURN
880 IF ta>tb THEN CLS : PRINT a$;"□HAS
    WON WITH□";ta;"□POINTS TO□";tb
890 IF tb>ta THEN CLS : PRINT b$;"□HAS
    WON WITH□";tb;"□POINTS TO□";ta
892 IF ta=tb THEN CLS : PRINT "THE
    RESULT IS A DRAW"
895 STOP
900 INPUT "ENTER THE PHRASE", LINE h$
910 IF h$<>s$ THEN PRINT FLASH 1;AT
    17,0;"WRONG!": PAUSE 50: LET
    tb=tb-50: PRINT INK 6; PAPER 2;AT
    1,22;tb:PRINT AT 15,0;"GUESS□";f:
    LET f=f+1: GOTO 360
920 FOR n=1 TO l: LET d$=z$(n): IF
    d$<>"*" THEN GOTO 950
930 LET m=(CODE s$(n)-64)*8-7: IF
    m=-263 THEN LET m=209
940 LET tb=tb+VAL q$(m+2 TO m+3)
950 NEXT n: GOTO 730
```

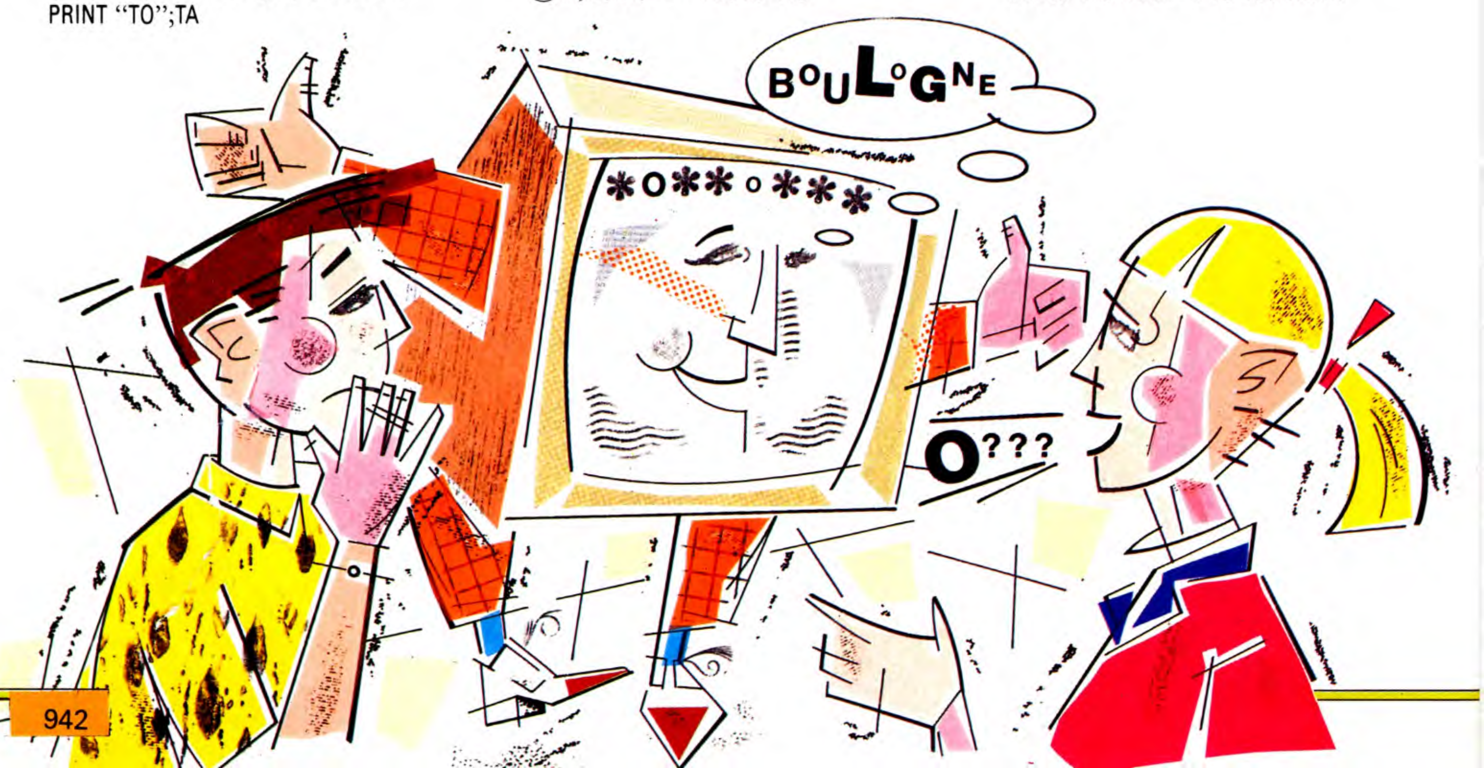
```
380 IF D$="" THEN
    M=131:D$="□":GOSUB 810:GOTO
    420
385 IF D$="ZZ" THEN D$="":GOTO 1000
390 IF D$="XX" THEN D$="":GOTO 500
400 IF D$<"A" OR D$>"Z" OR
    LEN(D$)>1 THEN GOSUB 2000:GOTO
    360
410 GOSUB 790
420 E=0
430 E=E+1
440 IF E=L+1 THEN TB=TB-G:Q$=
    LEFT$(Q$,M-1)+"□□□□"+
    MID$(Q$,M+4)
445 IF E=L+1 THEN PRINT "■■
    ■■■"Q$:D$="":GOTO 470
450 IF MID$(S$,E,1)<>D$ THEN 430
460 IF MID$(S$,E,1)=D$ THEN Z$=
    LEFT$(Z$,E-1)+D$+MID$(Z$,
    E+1):GOTO 430
470 GOSUB 950:GOSUB 2000
480 PRINT "■■■"TAB(20);
    TB"■□□":PRINT "■"LEFT$
    (QD$,11);Z$
485 PRINT LEFT$(QD$,13)SPC(15)
    "■■GUESS";F:F=F+1:IF S$=Z$
    THEN 730
490 GOTO 360
500 GOSUB 2000:PRINT LEFT$(QD$,
    21);:INPUT "GUESS AT WHICH
    CHARACTER";D$
510 IF LEN(D$)>1 THEN 500
520 ID D$=CHR$(32) THEN GOSUB 790:
    GOTO 550
530 IF D$<"A" OR D$>"Z" THEN 500
540 GOSUB 790
550 PRINT "■AT WHAT POSITION□-□
    USE CURSOR AND '0'■■■TO
    ENTER."
560 GOTO 660
570 GET Y$:IF Y$="" THEN 570
590 IF Y$="■" AND B<L-1 THEN
    B=B+1
600 IF Y$="■" AND B>0 THEN
```





```
   B=B-1
610 IF Y$="Ø" THEN 68Ø
66Ø PRINT "◆"LEFT$(QD$,11);Z$:
   PRINT "◯";TAB(B);"■■*■■"
67Ø GOTO 57Ø
68Ø IF MID$(Z$,B+1,1)<>D$ THEN 69Ø
685 TB=TB-G:PRINT "■■■"TAB
   (17)"CHEAT!■":FOR DE=1 TO
   1ØØØ:NEXT:B=Ø:GOTO 47Ø
69Ø IF MID$(S$,B+1,1)=D$ THEN 7ØØ
695 TB=TB-G/2:PRINT"■■■"TAB
   (15)"BAD LUCK!■":FOR DE=1 TO
   1ØØ:NEXT:B=Ø:GOTO 47Ø
7ØØ IF MID$(S$,B+1,1)<>D$ THEN 71Ø
7Ø5 PRINT "■■■"TAB(15)"GOOD
   GUESS■":FOR DE=1 TO 1ØØ:NEXT
7Ø6 Z$=LEFT$(Z$,B)+D$+MID$
   (Z$,B+2)
7Ø8 TB=TB+G:B=Ø
71Ø IF S$=Z$ THEN 73Ø
72Ø GOTO 47Ø
73Ø PRINT "■■■CONGRATULATIONS,
   □";B$;:GOSUB 95Ø:PRINT "◻■"
74Ø K=K+1:IF K=T*2 THEN 88Ø
75Ø C$=A$:A$=B$:B$=C$
76Ø TC=TA:TA=TB:TB=TC
77Ø Q$="":D=Ø:F=1
78Ø GOTO 16Ø
79Ø M=(ASC(D$)-64)*5-4
81Ø IF MID$(Q$,M,4)="□□□" THEN
   36Ø
82Ø G=VAL(MID$(Q$,M+2,2))
83Ø RETURN
88Ø IF TA>TB THEN PRINT "◻"A$;
   "□HAS WON WITH";TA;"POINTS":
   PRINT "TO";TB
89Ø IF TB>TA THEN PRINT "◻"B$;
   "□HAS WON WITH";TB;"POINTS":
   PRINT "TO";TA
892 IF TA=TB THEN PRINT "◻THE RESULT
   IS A DRAW"
895 END
95Ø FOR DE=1 TO 15ØØ:NEXT DE:
   RETURN
1ØØØ GOSUB 2ØØØ:PRINT LEFT$(QD$,
   2Ø)"■ENTER THE PHRASE□
   -■":INPUT GU$
1ØØ5 IF GU$=S$ THEN 1Ø2Ø
1Ø1Ø PRINT "■■■"TAB(17)"WRONG!
   ■":TB=TB-5Ø:PRINT "■■■"
   TAB(2Ø);TB"▯□□":GOSUB 95Ø
1Ø15 GOSUB 2ØØØ:PRINT LEFT$(QD$,
   13)SPC(15)"■■GUESS";F:
   F=F+1:GOTO 36Ø
1Ø2Ø FOR N=1 TO L:D$=MID$(Z$,
   N,1):IF D$<>"*" THEN 1Ø5Ø
1Ø3Ø M=(ASC(MID$(S$,N,1))-64)
   *8-7:IF M=-263 THEN M=2Ø9
1Ø4Ø TB=TB+VAL(MID$(Q$,M+2,2))
1Ø5Ø NEXT N:GOTO 73Ø
2ØØØ PRINT LEFT$(QD$,2Ø);SP$;
   SP$;SP$;SP$;SP$;"■■■";SP$;
   "▯□■■":RETURN
```

```
37Ø IF D$<>"XX" AND D$<>"ZZ" AND
   LEN(D$)>1 THEN 36Ø
38Ø IF D$=CHR$(32) THEN 41Ø
385 IF D$="ZZ" THEN D$="":GOTO 1ØØØ
39Ø IF D$<"A" OR D$>"Z" THEN 36Ø
4ØØ IF D$="XX" THEN D$="":GOTO 5ØØ
41Ø GOSUB 79Ø
42Ø E=Ø
43Ø E=E+1
44Ø IF E=L+1 THEN TB=TB-G:MID$(Q$,
   M,8)="□□□□□□□□":PRINT
   @96,Q$:D$="":GOTO 47Ø
45Ø IF MID$(S$,E,1)<>D$ THEN 43Ø
46Ø IF MID$(S$,E,1)=D$ THEN MID$
   (Z$,E,1)=D$:GOTO 43Ø
47Ø GOSUB 95Ø:PRINT@448:PRINT
   @416:PRINT@48Ø,STRING$(31,32);
48Ø PRINT@54,TB:PRINT@352,Z$:PRINT
   @32Ø,"GUESS";F:F=F+1:IF S$=Z$
   THEN 73Ø
49Ø GOTO 36Ø
5ØØ PRINT@448:PRINT@416:PRINT@
   448,"";:LINE INPUT "GUESS AT WHICH
   CHARACTER? ";D$
51Ø IF LEN(D$)>1 THEN 5ØØ
52Ø IF D$=CHR$(32) THEN GOSUB 79Ø:
   GOTO 55Ø
53Ø IF D$<"A" OR D$>"Z" THEN 5ØØ
54Ø GOSUB 79Ø
55Ø PRINT@448,"AT WHAT POSITION—
   USE ARROWS□□□□AND 'Ø' TO
   ENTER.";
56Ø PRINT@352,Z$:POKE 1Ø24+352+B,
   (ASC(MID$(Z$,B+1,1))AND 191)
57Ø Y$=INKEY$:IF Y$="" THEN 57Ø
59Ø IF Y$=CHR$(9) AND B<L-1 THEN
   B=B+1
6ØØ IF Y$=CHR$(8) AND B>Ø THEN
   B=B-1
61Ø IF Y$="Ø" THEN 68Ø
66Ø PRINT@352,Z$:POKE 352+1Ø24+B,
   (ASC(MID$(Z$,B+1,1))AND191)
67Ø GOTO 57Ø
68Ø PRINT@48Ø,STRING$(31,32);:
   IF MID$(Z$,B+1,1)=D$ THEN
   TB=TB-G:PRINT@448,"CHEAT!":
   FORDE=1TO1ØØ:NEXT:B=Ø:GOTO 47Ø
69Ø IF MID$(S$,B+1,1)<>D$ THEN
   TB=TB-G/2:PRINT@448,"BAD
   LUCK!":FOR DE=1 TO 1ØØ:NEXT:
```



```
   B=Ø:GOTO 47Ø
7ØØ IF MID$(S$,B+1,1)=D$ THEN
   PRINT@448,"GOOD GUESS":FOR
   DE=1 TO 1ØØ: NEXT:MID$(Z$,
   B+1,1)=D$:TB=TB+G:B=Ø
71Ø IF S$=Z$ THEN 73Ø
72Ø GOTO 47Ø
73Ø PRINT@48Ø,"CONGRATULATIONS,
   □";B$;:GOSUB 95Ø:CLS
74Ø K=K+1:IF K=T*2 THEN88Ø
75Ø C$=A$:A$=B$:B$=C$
76Ø TC=TA:TA=TB:TB=TC
77Ø Q$="":D=Ø:F=1
78Ø GOTO 16Ø
79Ø M=(ASC(D$)-64)*8-7
8ØØ IF M=-263 THEN M=2Ø9
81Ø IF MID$(Q$,M,6)="□□□□□□"
   THEN 36Ø
82Ø G=VAL(MID$(Q$,M+2,2))
83Ø RETURN
88Ø IF TA>TB THEN CLS:PRINT A$;
   "□HAS WON WITH";TA;"POINTS":
   PRINT"TO";TB
89Ø IF TB>TA THEN CLS:PRINT B$;
   "□HAS WON WITH";TB;"POINTS":
   PRINT"TO";TA
892 IF TA=TB THEN CLS:PRINT"THE RESULT
   IS A DRAW"
895 END
95Ø FOR DE=1 TO 15ØØ:NEXTDE:RETURN
1ØØØ PRINT@448:PRINT@416:PRINT@
   416,"ENTER THE PHRASE— ":LINE INPUT
   GU$
1Ø1Ø IF GU$<>S$ THEN PRINT@416,
   "WRONG!":TB=TB-5Ø:PRINT@54,
   TB:GOSUB 95Ø:PRINT@32Ø,
   "GUESS";F;:F=F+1:GOTO 36Ø
1Ø2Ø FOR N=1 TO L:D$=MID$(Z$,
   N,1):IF D$<>"*" THEN 1Ø5Ø
1Ø3Ø M=(ASC(MID$(S$,N,1))-64)
   *8-7:IF M=-263 THEN M=2Ø9
1Ø4Ø TB=TB+VAL(MID$(Q$,M+2,2))
1Ø5Ø NEXT N:GOTO 73Ø
```

As with the last part of the word game, all the programs are very similar, except the one for the Acorn machines.

Lines 37Ø to 41Ø handle the guesser's input—the choices of buying letters or guessing. D$ (or d$) is the guesser's decision, and the routine checks the input to see whether it is just a single letter or a space—to signify buying—or if it is XX, or ZZ, for a guess. The routine also traps any invalid inputs by sending the program back to the input prompt line—Line 36Ø

BUYING LETTERS

If a player decides to buy a letter, the first thing the computer does is to check what value is placed on it. To do this, the program sends it on to the subroutine starting at Line 79Ø.

This checks the ASCII value of the letter, then performs a calculation on this value to find out how far through the table of letter values this occurs. If the computer finds a blank at that point, it means that the chosen letter has already been bought, so the program goes back to Line 36Ø for the player to pick again. Otherwise, it then slices out the part of the string containing the value, which it assesses using VAL. This will be used in a moment to update the player's score.

The routine which looks after letter buying starts at Line 43Ø. Lines 43Ø to 46Ø step through the phrase, looking for occurrences of the bought letter. The dummy string is updated, replacing the asterisks with the letter in every position that it occurs in the phrase and the new string is then displayed. Its value is subtracted from the guesser's total whether the letter occurs in the phrase or not. The running total of guesses, F (or f) is incremented. Line 44Ø overprints the chosen letter and its value with blanks to signify that this option is no longer available.

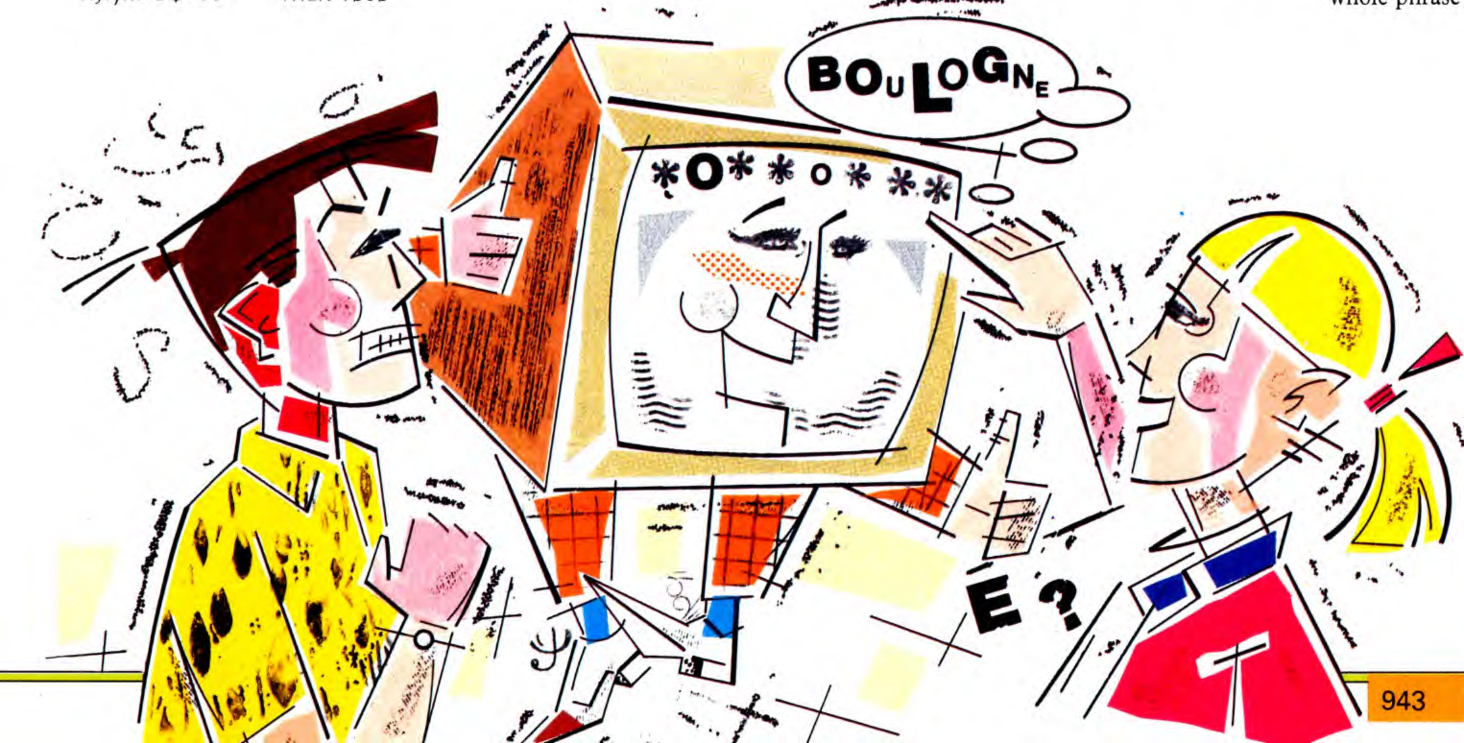
SPECIFIC LETTERS

If the guesser wishes to try a specific letter in a specific position, XX will be selected. This causes the program to jump to Line 5ØØ. The guesser must first enter the letter to be tried. There are a number of error traps to ensure that the input is legal. If a legal choice is made, then the guesser is given instructions about positioning the cursor to insert the letter. With the cursor positioned at the correct place, the Ø key has to be pressed. The program makes sure the choice is allowable, and checks if it is correct.

If the letter is wrong, Line 69Ø tells the guesser BAD LUCK, and subtracts half the letter's value from the guesser's score as a penalty. If the letter and position are correct, then Line 7ØØ tells the guesser GOOD GUESS, and adds the letter's value to the score. If the phrase has now been completed, Line 71Ø sends the program to Line 73Ø, which announces to the lucky guesser CONGRATULATIONS.

THE WHOLE PHRASE

If the guesser is more ambitious, and has selected the option to guess the whole phrase



(ZZ), then Line 385 sends the program to Line 9ØØ, in the case of the Spectrum program, and Line 1ØØØ in the case of the Commodore and Dragon/Tandy programs. The routine asks the guesser to enter the phrase. The guess is compared with the phrase originally entered. If the guess is wrong, 5Ø points are deducted from the score, and the guess count is incremented. If the guess is correct, then the score for the unguessed letters is calculated, and the program jumps back to Line 73Ø, which displays CONGRATULATIONS.

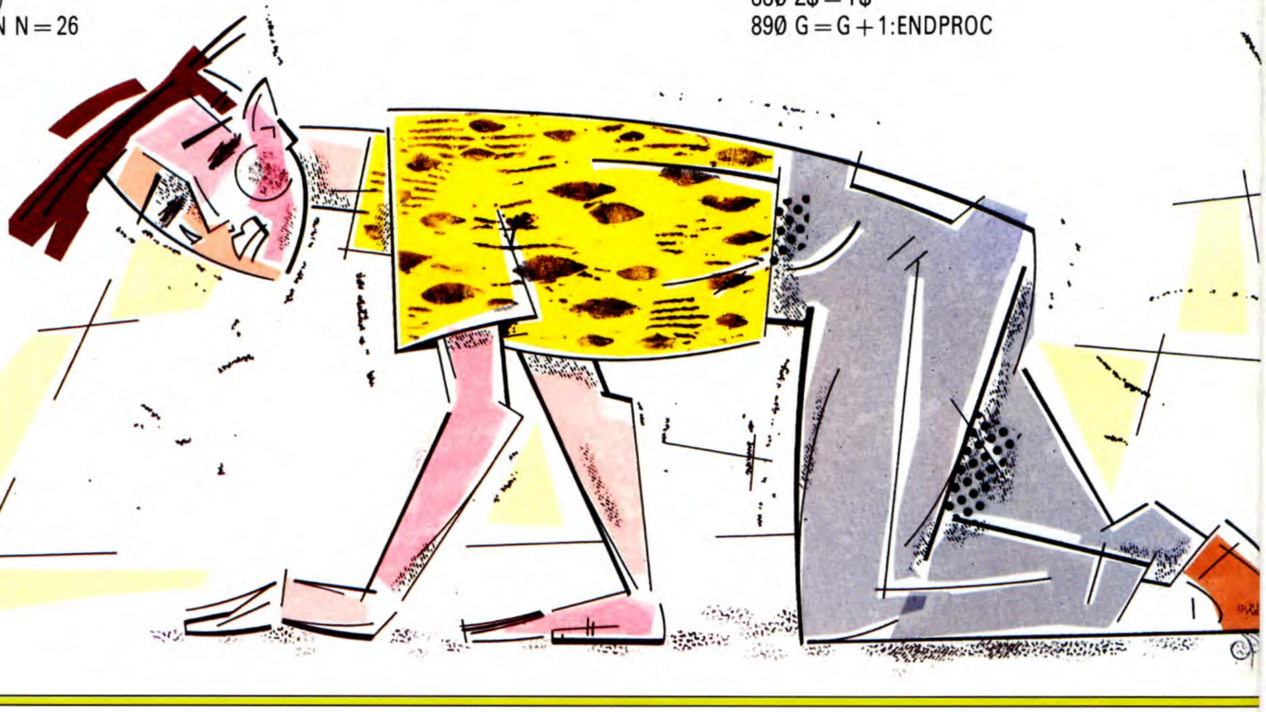
THE END

After the phrase has been guessed correctly, the program checks if the number of goes for each player chosen at the start of the game have been used up. If they have not, the turn passes to the next player, after the strings and variables used in the game have been reset.

If the game has finished, then the program jumps to Line 88Ø. The scores are compared in this and the following two lines, and the final outcome of the game displayed.

The game ends here, but you may wish to add an 'another go?' routine to make the game really complete.

```
90 PROCINPUT
100 P=3-P
110 IF TP=2 AND TQ=NG THEN 130
120 PROCWORD
130 NEXT
140 NEXT
150 GOTO 920
240 DEF PROCKILL(N)
250 IF N=-33 THEN N=26
260 PRINTTAB((N MOD 4)*10,4+N DIV
   4)SPC10
270 B$=B$+CHR$(N+65)
280 ENDPROC
290 DEF PROCINPUT
300 PRINTTAB(Ø,2)"□□□□"; S(1)"□"
   TAB(23,2)S(2)"□"TAB(Ø,16)Z$:
   IF G<>Ø THEN PRINTTAB(15,15)
"GUESS ";G
310 PRINTTAB(Ø,16)Z$:IF G<>Ø THEN
   PRINTTAB(15,15)"GUESS□";G
320 VDU23,1,1;Ø;Ø;Ø;
330 IF INSTR(Z$,"*")=Ø THEN
   PRINTTAB(Ø,18)"CONGRATULATIONS
   □□□":FOR T=1 TO 10000:
   NEXT:ENDPROC
340 VDU 31,X-1,16:K=GET:VDU23,
   1,Ø;Ø;Ø;Ø;
350 IF K=136 AND X>1 THEN X=X-1:
   GOTO 300
360 IF K=137 AND X<LENZ$ THEN
   X=X+1:GOTO 300
370 IF K=2 THEN PROCBUY
380 IF K=7 THEN PROCGUESS
390 IF K<>32 AND (K<65 OR K>90)
   THEN 300
400 IF MID$(Z$,X,1)<>"*" THEN 300
410 IF INSTR(B$,CHR$K)<>Ø THEN 300
420 G=G+1
430 PS=Ø:IF MID$(Y$,X,1)=CHR$K THEN
   PS=1
440 IF PS=Ø THEN S(3-P)=S(3-P)-
   V(K-64-(K=32)*59)/2:PRINTTAB
   (Ø,18)"BAD LUCK":FOR T=1 TO
   3000:NEXT:PRINTTAB(Ø,18)SPC(8):
   GOTO 300
450 S(3-P)=S(3-P)+V(K-64-
   (K=32)*59):Z$=LEFT$(Z$,
   X-1)+CHR$(K)+MID$(Z$,X+1)
460 GOTO 300
710 DEF PROCBUY
720 PRINTTAB(Ø,18)"BUYING A
   CHARACTER"
730 K=GET:IF (K<65 OR K>9Ø) AND
   K<>32 THEN 730
740 IF INSTR(B$,CHR$K) THEN 780
750 FOR T=1 TO L:IF MID$(Y$,T,1)
   =CHR$K THEN Z$=LEFT$(Z$,T-1)
   +CHR$K+MID$(Z$,T+1)
760 NEXT
770 B$=B$+CHR$K:PROCKILL(K-65):
   S(3-P)=S(3-P)-V(K-64-(K=32)
   *59):G=G+1
780 PRINTTAB(Ø,18)SPC18:ENDPROC
790 DEF PROCGUESS
800 PRINTTAB(Ø,18)"GUESS THE PHRASE"
810 INPUT""A$
820 IF LENA$<>L THEN PRINTTAB
   (Ø,18)"WRONG LENGTH□□□□":
   PRINTSTRING$(8Ø,"□"):FOR
   T=1 TO 3000:NEXT:GOTO 800
830 IF A$<>Y$ THEN PRINTTAB
   (Ø,18)"THAT IS WRONG□□□":
   PRINTSTRING$(8Ø,"□"):S(3-
   P)=S(3-P)-5Ø:ENDPROC
840 FOR T=1 TO L
850 IF INSTR(B$,MID$(A$,T,1))
   THEN 870
860 IF MID$(Z$,T,1)="*" AND
   MID$(Y$,T,1)=MID$(A$,T,1)
   THEN S(3-P)=S(3-P)+V(ASC(MID$
   (A$,T,1))-64-(MID$(A$,T,1)
   ="□")*59)
870 NEXT
880 Z$=Y$
890 G=G+1:ENDPROC
```



```
920 IF S(1)<S(2) THEN T=S(1):
    S(1)=S(2):S(2)=T:A$=A$(1):
    A$(1)=A$(2):A$(2)=A$
930 CLS:PRINTTAB(12,10)
    "FINAL RESULTS"
940 IF S(1)=S(2) THEN PRINTTAB
    (0,13)"IT WAS A DRAW AT□";
    S(1);"□POINTS EACH":END
950 PRINTTAB(0,13)A$(1)"□BEAT□"
    A$(2)"□BY□";S(1);"□TO□";
    S(2);"□POINTS"
```

The Acorn program presents the options to the guesser slightly differently from the other programs, but the game is exactly the same.

This is how the program works:

INPUT

Line 90 calls PROCINPUT, which extends from Line 290 to Line 460. There is a prompt to input a letter which the guesser wishes to place in a specific position in the phrase. This is done by pressing the left and right cursor keys, but at this stage the guesser can select the buy or guess a phrase option instead. If either of these are chosen, the program jumps to PROCBUY or PROCGUESS.

The remainder of PROCINPUT checks whether the input is correct, and tells the guesser the outcome.

BUYING LETTERS

PROCBUY, starting at Line 710 allows the guesser to buy a character. The PROCedure checks if the chosen letter is valid before proceeding to check where (if at all) the letter occurs in the phrase to be guessed.

If the letter is present in the phrase or not, the end section of Line 770 subtracts the value of the letter from the guesser's total, and increments the number of guesses.

PROCKILL blanks out the bought letter from the table.

GUESSING THE PHRASE

PROCGUESS, starting at Line 790, prompts the guesser for the phrase to be tried. The length and correctness of the phrase are checked, and the guesser is told the outcome. If the phrase is wrong, the guesser loses 50 points.

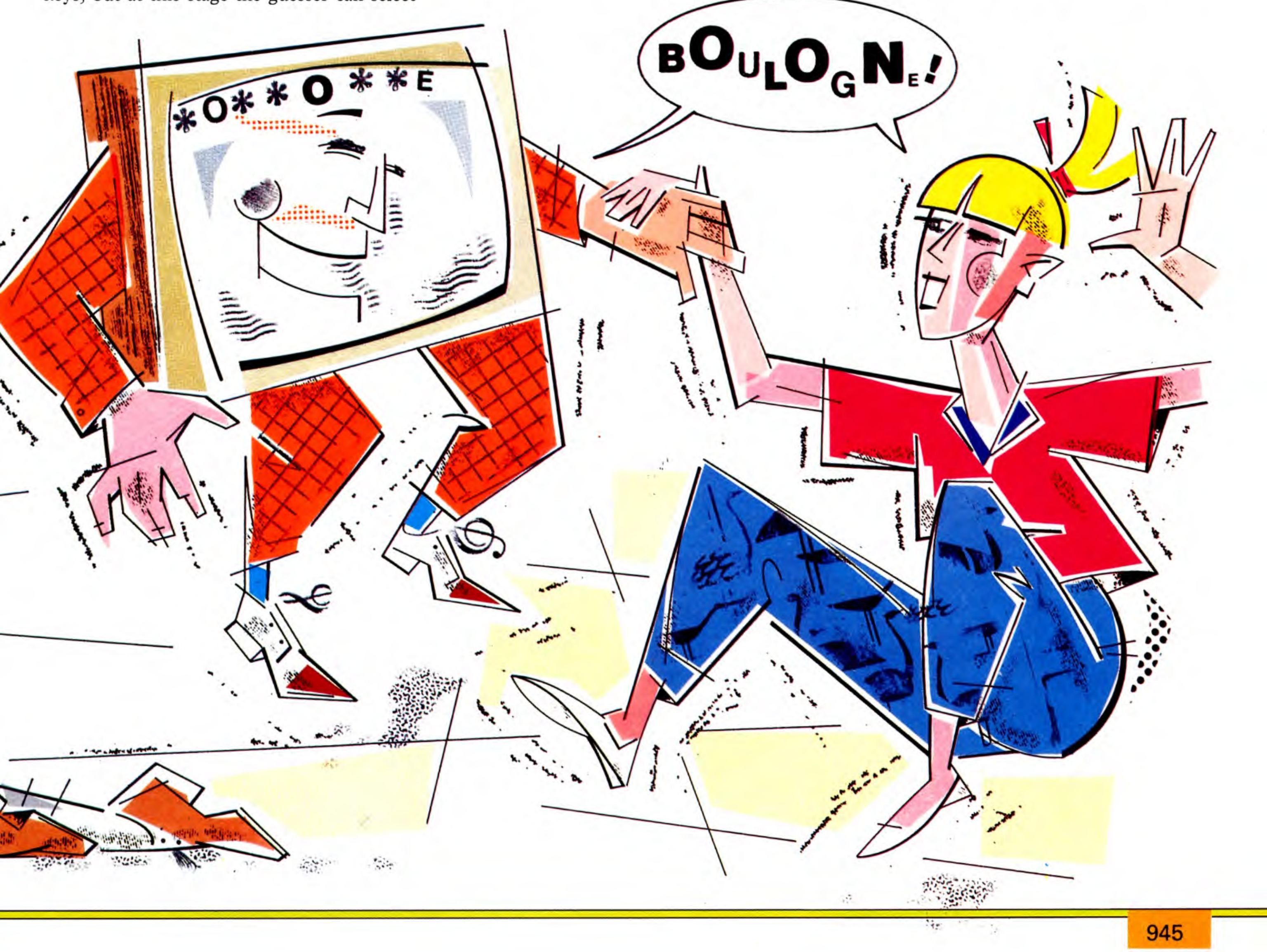
If the phrase is correct, Lines 840 to 870 calculate the value of the letters which were not displayed at the time the phrase was correctly guessed and adds this total to the guesser's score.

COMPLETING THE GAME

After PROCINPUT has been completed, the program continues from Line 100. It's now the guesser's turn to input a phrase, but first, Line 110 checks if the number of turns chosen at the start of the game have been completed. If they have, the program jumps to Line 920.

The 'end of game' routine displays the winner if there was one, or informs the players of a draw.

Finally, you may wish to add an 'another go?' routine to the program to round it off.



EXTEND YOUR HOBBIES FILE

Here are some extra routines to add to your hobbies file program to make it even more useful, and you can choose which ones to add to suit your exact needs



- EXTENDING THE FILE
- ADDING NEW OPTIONS
- UPDATING THE ROUTINES
- CONTINUOUS PRINTOUT
- A NEW SORT
- SEARCHING MORE THAN ONE FIELD AT A TIME
- ADDING A NEW FIELD TO AN EXISTING FILE
- CONVERTING TO DISK

The datafile program given on pages 46 to 53 and 75 to 79 is an extremely useful way of storing information. It doesn't matter whether it's details of your hobby, information from a survey, a mailing list for your business, or any other mass of data. If you put it into the datafile you can look up records, amend them, print them out, delete some and still have an ordered list at the end of it.

However, no general purpose datafile can possibly suit all possible applications. This time, so you can tailor your datafile to meet your specific needs, there are some extra routines to add to the program and other routines to update existing sections. As usual the extra routines are different for each computer and they are described below separately for each machine.

The extra routines for the Spectrum are a continuous printout option, a continuous input of records and a conversion for the Microdrive.

CONTINUOUS INPUT

By entering these lines you can input your records one after the other without having to return to the menu each time. Press ENTER when you've entered the records. These lines also prevent you entering a null string for the first record (which would confuse the program). If you do enter a null string the program returns to the main menu instead:

```
2ØØØ CLS : LET C=V
211Ø FOR N=V TO A: PRINT INVERSE V;AT
   V+N*2,Ø;N$(N);AT V+N*2,12; FLASH
   V;"?": INPUT "(up to□";A(N);"□
   characters)",LINE A$(C,B(N)+V
   TO B(N+V)): IF N=V AND A$(C,B(N)
   +V)=CHR$ 32 THEN RETURN
2115 PRINT AT V+N*2,12; A$(C,B(N)+V
   TO B(N+V)): NEXTN
212Ø FOR F=V TO 15Ø: NEXT F: IF C=V
   THEN GOTO 2ØØØ
214Ø IF A$(C)>=A$(C−V) THEN GOTO
   2ØØØ
215Ø LET X$=A$(C): LET A$(C)=
   A$(C−V): LET A$(C−V)=X$: LET C=
   C−V: IF C=V THEN GOTO 2ØØØ
```

CONTINUOUS PRINT

At present the print option allows you to print out only one record at a time. If you want to print out the whole list you have to sit by the computer pressing F to go on to the next record. The continuous print option prints out all records, from the one you are looking at to the end. It can be stopped at any time by pressing any key. If you want to print out all records, then just make sure you start at record 1.

```
12Ø LET OP=1
3Ø15 IF D−V=R THEN LET D=D−V: IF
   OP=6 THEN LET OP=1
3Ø2Ø IF A$(D,V)=CHR$ 32 THEN LET
   D=D−V: IF OP=6 THEN LET OP=1
3Ø85 IF OP=6 THEN LET D=D+V: GOTO
   3Ø1Ø
4Ø6Ø IF D>R THEN LET D=PM: IF OP=6
   THEN LET OP=1
4Ø8Ø IF A$(D,V)=CHR$ 32 THEN LET
   D=PM: IF OP=6 THEN LET OP=1
4165 IF OP=6 THEN LET MO=V: LET
   D=D+MO: GOTO 4Ø6Ø
95Ø2 IF OP=6 AND INKEY$="" THEN
   COPY : RETURN
95Ø5 PRINT INVERSE V;AT
   19,U;"□C(ontinuous print)";TAB 31;"□"
9585 IF V$="C" THEN COPY : LET OP=6
```

MICRODRIVE CONVERSION

These lines allow the program to work with the Microdrive. Don't type them in if you are using the program with tape as they overwrite the old ones. If you are upgrading from tape to Microdrive, LOAD in the old program and your existing data first, break into it and add these lines, then SAVE the new program and the data that it contains to the Microdrive.

```
5ØØØ CAT 1
5ØØ5 INPUT "Enter file name to save under",
   LINE Q$: IF LEN Q$<V OR LEN Q$>1Ø
   THEN GOTO 5ØØ5
5Ø1Ø SAVE *"M",1;Q$ LINE 1Ø: VERIFY
   *"M",1;Q$: RETURN
6ØØØ CAT 1
6Ø1Ø INPUT "Enter name of file to be loaded",
   LINE X$: IF LEN X$<V OR LEN X$>1Ø
   THEN GOTO 6Ø1Ø
6Ø2Ø PRINT #1;"LOADING FILE NOW":
   LOAD *"M",1;X$
```

There are six extra routines for the Commodore. These are a continuous print option, a new search and sort, a facility to add extra fields to an existing file, a disk load and save routine and a verify option.

CONTINUOUS PRINT

The new lines allow you to print out all the records in the file with one keypress—C—for continuous print. It saves you having to press F for forward then P for print each time. The printout starts from the record you are viewing at the time so if you want a list of all your records be sure to start from the beginning.

```
313Ø printcu$cu$cu$rt$x3$x2$;
   spc(11)"C-print";
314Ø ifaa$="c"then363Ø
32ØØ ok$="fbmadp□c":gosub1Ø6ØØ
33ØØ printro$;:onbgoto39ØØ,3ØØØ,
   199Ø,37ØØ,34ØØ,36ØØ,39ØØ,36ØØ
365Ø next:print#4,ul$:ifb=8or
   aa$="c"thenaa$="c":b=1:
   goto39ØØ
366Ø close4:goto31ØØ
372Ø y=sh−2+2*vic:x=Ø:gosub115ØØ:
   printro$;:z=ll*(2−2*vic)−2:
   gosub135ØØ
3722 ifb=6orb=8thenreturn
39Ø2 ifaa$="c"thenclose4:aa$="n"
```

SORT ANY FIELD

At present the records are sorted automatically on field 1 as you enter them. The new lines let you reorganize the data so it can be sorted on any of the fields. You can do this at any time by choosing option 8 from the main menu and entering the new field number. This field is then displayed at the top.

```
6 dimfc(8,1):fori=1to8:readfc(i,Ø):fc(i,1)=
   −1:next:data−1,,,,,−1,−1,
52 cl$=chr$(157):cr$=chr$(29):
   c4$=cd$+cd$+cd$
76 dimm$(9):fori=Øto9:readm$(i):
   next:data"□Resequence data"
558 printtab(1Ø)cu$x1$"7"x2$m$(7)
   "[illegible]"
```

```
560 if(nf>1)andfdthenprinttab(9)
    x1$cu$cu$"8"x2$m$(Ø)x3$;cd$cd$
620 a=asc(a$)-48:ifa<1ora>8then
    gosub1ØØØØ:goto65Ø
67Ø a=asc(a$)-48:ifa<1ora>8then
    gosub1ØØØØ:goto6ØØ
89Ø ifu=Øand((a=3)or(a=4)or(a=8))
    thengosub1ØØØØ:goto6ØØ
9ØØ onagosub1ØØØ,2ØØØ,398Ø,4ØØØ,
    95Ø,95Ø,7ØØØ,8ØØØ
95Ø if aa$="n" then clr:gosub6:a=6:
    gosub955:goto1ØØ
955 printro$;cs$;gr$;tab(11);m$(a)
    c4$"■"
8ØØØ printcd$"□□□□PLEASE INPUT
    FIELD NUMBER□"x3$;
8Ø1Ø ok$=left$("2345678",nf-1):
    gosub1Ø6ØØ:printx2$;(ix+1);
8Ø2Ø ix$=hd$(ix+1):hd$(ix+1)
    =hd$(1):hd$(1)=ix$:i=lx
    (ix+1):lx(ix+1)=lx(1)
8Ø25 lx(1)=i
8Ø3Ø i=hx(ix+1):hx(ix+1)=
    hx(1):hx(1)=i
8Ø4Ø forup=Øtou-1:ix$=t$(up,ix):
    t$(up,ix)=t$(up,Ø):t$(up,Ø)=ix$:next
8Ø5Ø dn=u:ifu=1thenreturn
8Ø6Ø dn=int(dn/2+.6):i=Ø:
    forup=dntou-1
8Ø7Ø ift$(r(up-dn),Ø)>t$
    (r(up),Ø)theni=1:ru=r(up-
    dn):r(up-dn)=r(up):r(up)=ru
8Ø8Ø next:dn=dn+i:if(dn>1)then8Ø6Ø
8Ø9Ø return
```



MULTIPLE SEARCH

This option allows you to search more than one field at once. Choose the search option from the main menu then enter the number of the first field to be searched and the entry you're looking for. If you are searching more than one field answer Y to 'Anything else?' and enter details of the other fields. You are then asked if all of these must be found together. If you answer N then the program will print out all records with at least one of the things you are looking for. If you answer Y, the program will print out records with all items you're searching for.

For example, if you had a mailing list on file and were looking for someone called Smith who lived in London, answering Y to the question gives you all Smiths in London, while answering N gives you all Smiths wherever they live *and* all people who live in London.

If you now choose the Continuous print option it will print out the records found that satisfy these criteria, not all the records in the file.

```
3935 ifaa$="y"thengosub4ØØØ
4ØØØ fg=Ø
4ØØ2 printcs$x1$gr$"□WHICH FIELD IS TO
    BE SEARCHED?:"cc$"□"cl$;
4Ø3Ø printchr$(ix+48):fx(fg)=ix
4Ø8Ø fx$(fg)=ix$:ff(fg)=len(ix$)
4Ø9Ø printrt$x1$"□ANYTHING ELSE
    (y/n)?":gosub1Ø5ØØ:ifaa$="y"
    thenfg=fg+1:goto4ØØ2
4Ø92 iffg>Øthenprintrt$x1$"□MUST ALL
    THESE BE FOUND TOGETHER
    (y/n)?":gosub1Ø5ØØ
4Ø94 fh=(aa$="y")
411Ø for gf=Øtofg
4112 ix$=t$(r(up),fx(gf)-1)
412Ø fe=len(ix$)-ff(gf)+1
413Ø iffe<1then4154
414Øforj=1tofe:ifmid$(ix$,j,ff
    (gf))<>fx$(gf)thennext:goto
    4154
415Ø if(gf=fg)ornot(fh)then3Ø4Ø
4152 next gf
4154 ifnot(fh)thennextgf
```

ADD A NEW FIELD

This is a very useful facility that allows you to add extra fields to an existing datafile. Now, if you discover that you need to extend your file it is relatively easy to do so. With the original program you would have to re-enter all the data.

Before doing anything else, make sure you have a copy of your existing data on tape.

To set up the new file with the extra fields you'll need to know the structure of the old file—the number of fields along with their names and lengths. So look this up first. If you haven't made a note of these you'll have to LOAD in your existing data first to check. Now choose option 1 to create a new file and enter all the details for the old fields plus the new ones. But don't enter any data—hit RETURN instead for the main menu. Now choose option 6 to LOAD in your old data from tape and fill in the new fields one at a time using the Amend option.

```
83 data,"Merge tape file□"
802 ifa=6thenprinttab(11)x1$"MERGE
    EXISTING DATA?";:gosub
    10500:a=12-3*ix:goto900
900 onagosub1000,2000,3980,4000,
    950,950,7000,8000,950
980 ifa>=6then6000
6112 ifa=9then6500
6220 close1:goto6980
6500 input#1,u0,n0,ix
6510 ifu+u0>vthenprint"TOO
    MUCH TO MERGE, CUT SHORT";:
    u0=v-u:for n=1to1000:next
6520 forn=1ton0:input#1,ix$,
    ix,ix:next
6530 forup=0tou0:forn=1ton0:input
    #1,t$(up+u,n-1):next
6540 input#1,ru:r(up+u)=
    ru+u:next
6550 close1:ix=1:u=u+u0:
    goto8050
```

CHANGING TO DISK

Add these lines if you want to use the datafile with a disk drive. They overwrite the original routine that SAVEd and LOADed data from tape. (By the way, the original Line 6000 for tape load should read 'Position tape for output', not 'Position tape for input' as printed.)

```
5000 printcs$rv$"□□□□□
    □□□□INSERT DISK FOR OUTPUT
    □□□□□□□□□□"
5100 open 1,8,1,f$
6000 printcs$rv$"□□□□□□
    □□□INSERT DISK FOR INPUT
    □□□□□□□□□□"
6100 open 1,8,0,f$
6220 close 1:goto 6980
```

If you want to transfer your data to disk from tape, type in the new disk SAVE routine first (Lines 6000 to 6220), LOAD in the data from tape and SAVE to disk. Then type in the disk LOAD routine (Lines 5000 and 5100). You'll then be able to LOAD the new datafile on the disk as well.

VERIFY THE DATA

If you add these lines you'll be able to verify the DATA once it's SAVEd:

```
5160 print"□DO YOU WANT TO
    VERIFY? (y/n)":gosub10500:
      ifix=1then6000
6114 ifa=5then6700
6700 input#1,u0,n0,ix
6702 if(u0<>u)or(n0<>nf)or
    (ix<>tt)thenprint"FIRST ITEMS
    WRONG":goto6750
6710 forn=1tonf:input#1,ix$,n0,ix
6712 if ix$<>hd$(n)or n0<>lx(n)
    or ix<>hx(n)thenprint
    "WRONG HEADING";n: goto6750
6720 next:forup=0tou:forn=1
    tonf:input#1,ix$
6722 if ix$<>t$(up,n-1)thenprint
    "DATA WRONG rec";up;"field";n:
    goto6750
6730 next:input#1,ix
6732 if ix<>r(up)thenprint
    "POINTER WRONG rec",up:
    goto6750
6740 next:print"VERIFIED OKAY"
6750 close1:fori=1to2000:next: return
```

If you answer y to the question 'do you want to verify' you are invited to reposition the tape at the start of the DATA and the DATA is then verified producing an error message or VERIFIED OKAY.

Test the verify option by creating and SAVEing some data and verifying it. Then amend a data field. When the program offers you 'verify' the second time, rewind the tape, to check the new data in memory against the old data on tape. It will fail at the appropriate field of the record you altered.

If the program stops with an error message when you haven't altered any records type GOTO 100 followed by RETURN to get back into the program, and try SAVEing the DATA again.

One point to note is that you cannot use this option to bypass old data files in the same way that you might bypass old programs using the normal program verify. This is because, unlike the program verify, this one does not go all the way to the end of a file after finding an error. So take care to position the tape carefully at the beginning of the correct data file.

There are five new routines for the Acorn computers. These are continuous print, new search and sort options, the facility to add extra fields to an existing file, and changes for a disk drive.

CONTINUOUS PRINT

The new lines give you the option of printing out all the records in one go. Press P for print as usual, then answer Y to the question Do you want all your records?

```
145 IF G<>7 THEN OT=G:GOTO 148
9004 IF G=67 THEN VDU2
9015 PRINT
9025 IF G=67 THEN VDU3
9500 DEF PROCPRINTER
9510 PRINT'"Check printer—
    C(ontinue)":G=GET AND &5F:
    IF G<>67 THEN VDU11:PRINT
    STRING$(39,"□"):VDU11,11:
    ENDPROC
9520 PRINT'"DO YOU WANT ALL YOUR
    RECORDS (Y/N)":G=GET AND &5F:IF
    G<>89 THEN C=0:PM=D%:GOTO
    9540
9530 D%=0:C=1:F=1:PM=N%-1:
    Q=0
9540 REPEAT
9550 IF OT=3 THEN D%=D%+C ELSE
    PROCFIND
9560 IF Q=2 THEN 9580
9570 G=67:PROCVDU
9580 VDU2,1,10,1,10,3
9590 UNTIL PM=D%
9600 ENDPROC
```

SORT ANY FIELD

At present the records are sorted automatically, on field 1, as you enter them. The new routine lets you reorganize the data so it is sorted on any of the other fields. You can do this at any time by choosing option 8 from the main menu and entering the number of the field you want sorted. The fields are displayed in the same order as before (so if it was a mailing list the name would still be on top) but the records are sorted in order based on the new control field (street, town etc.).

```
1 MODE7:M%=0:N%=1:CF=1
4 HIMEM=PAGE+&3000
5 DIMA(8),N$(8),TRL(8),A$(8),N(8)
117 PRINT'"□□□□□□□8 :—Reorganize
    Records"
130 G=GET-48:IF G<1 OR G>8 THEN
    130
140 IF M%=0 AND (G>1 AND G<6 OR
```



```
   G=8) THEN 13Ø
148 ON G GOTO 15Ø,16Ø,17Ø,18Ø,19Ø,
   2ØØ,Ø,21Ø
21Ø PROCSORT:GOTO 3Ø
2Ø9Ø X=B%+G*R%:Y=B%+(G-1)*
   R%:IF $(X+TRL(CF))>=$(Y+TRL
   (CF)) THEN 2ØØØ
31ØØ IF $(X+TRL(CF))>=$(Y+TRL
   (CF)) THEN313Ø
1ØØØØ DEF PROCSORT
1ØØ1Ø CLS:IF A=1 THEN PRINT"CAN'T
   REORGANIZE ONE FIELD":FOR T=1 TO
   3ØØØ:NEXT:ENDPROC
1ØØ2Ø PRINT"FIELD NO.","NAME"
1ØØ3Ø FOR N=1 TO A:PRINT;N,N$(N):
   NEXT
1ØØ4Ø PRINT:PRINT"ENTER NEW CONTROL
   FIELD NO. (1 TO□";A;")"
1ØØ5Ø INPUTAA:AA=INT(AA):IF AA<1 OR
   AA>A THEN PRINT"RE-ENTER":GOTO
   1ØØ5Ø
1ØØ6Ø CF=AA
1ØØ7Ø CLS:PRINT"REORGANIZING FILE"
1ØØ8Ø FOR N=1 TO N%-2:K=N
1ØØ9Ø FOR J=N+1 TO N%-1
1Ø1ØØ X=B%+J*R%:Y=B%+K*R%
1Ø11Ø IF $(X+TRL(CF))<$(Y+TRL
   (CF)) THEN K=J
1Ø12Ø NEXT:IF N=K THEN 1Ø15Ø
1Ø13Ø X=B%+N*R%:Y=B%+K*R%
1Ø14Ø FOR T=1 TO A:$B%=$(X+TRL
   (T)):$(X+TRL(T))=$(Y+TRL(T)):
   $(Y+TRL(T))=$B%:NEXT
1Ø15Ø NEXT:ENDPROC
```

MULTIPLE SEARCH

This option allows you to search more than one field at once. Choose the usual search option from the menu—option 4—then enter the number of fields you want to search. You are then asked if you want an AND or an OR search. An AND search means that all the things you are looking for have to be found together before a record is printed out. An OR search will print out a record as long as it contains at least one of the things you are looking for.

As an example, say you had a stamp collection on file and you were searching for British stamps in field 2 and 10p stamps in field 3. An AND search prints out all British 10p stamps, while an OR search prints out all British stamps and all 10p stamps.

If you now choose Continuous print it will print out all records found, not all the records on the file.

Delete Lines 5Ø1Ø to 5Ø9Ø of the original program then add:



```
5Ø1Ø CLS:IF N%=1 THEN ENDPROC
5Ø2Ø INPUT'"HOW MANY FIELDS DO YOU
   WANT TO SEARCH",NMF
5Ø3Ø IF NMF=Ø THEN ENDPROC
5Ø4Ø IF NMF<Ø OR NMF>A THEN 5Ø2Ø
5Ø5Ø IF NMF=1 THEN TYPE=Ø:GOTO 5Ø75
5Ø6Ø INPUT"DO YOU WANT (A)ND OR (O)R
   SEARCHES",A$
5Ø7Ø IF A$<>"A" AND A$<>"O" THEN
   5Ø6Ø ELSE TYPE=-(A$="A")
5Ø75 CLS:PRINT"FIELD NO.□□□",
   "NAME"':FOR T=1 TO A:PRINT
   '"□□□□";T,"□",N$(T):
   NEXT:PRINT
5Ø8Ø FOR P=1 TO NMF
5Ø9Ø INPUT"ENTER SEARCH FIELD
   NUMBER",N(P)
51ØØ IF N(P)<Ø OR N(P)>A THEN
   PRINT"RE-";:GOTO 5Ø9Ø
511Ø PRINT"LOOK FOR WHAT ? ";:
   PROCINPUT(A(P)):A$(P)=$B%:PRINT
512Ø NEXT
513Ø D%=Ø:C=1
514Ø F=Ø:PM=D%+C:IF PM=Ø THEN
   PM=N%-1
515Ø IF PM=N% THEN PM=1
516Ø PROCFIND
517Ø IF Q=2 THEN 52ØØ
518Ø Q=Ø:PROCVDU:PROCKEY:IF Q=1
   THEN ENDPROC
```

```
519Ø GOTO 514Ø
52ØØ CLS:PRINTTAB(Ø,1Ø)"THERE ARE NO
   RECORDS WITH THAT SPEC."
521Ø FOR P=1 TO 5ØØØ:NEXT
522Ø ENDPROC
523Ø DEF PROCFIND
524Ø Q=Ø:LOCAL T
525Ø REPEAT
526Ø D%=D%+C:IF D%=Ø THEN
   D%=N%-1
527Ø IF D%=N% THEN D%=1
528Ø FOR T=1 TO NMF
529Ø IF A$(T)<>$(B%+D%*R%+TRL
   (N(T))) THEN 532Ø
53ØØ IF TYPE=Ø THEN T=NMF
531Ø IF T=NMF THEN Q=1:NEXT:
   GOTO 535Ø ELSE 533Ø
532Ø IF TYPE=1 THEN T=NMF
533Ø NEXT
534Ø IF F=1 AND PM=D% THEN Q=2
   ELSE F=1
535Ø UNTIL Q<>Ø
536Ø ENDPROC
```

ADD A NEW FIELD

This allows you to add new fields to an existing file—ideal when you find you need to extend the range of your data or find you've forgotten some important aspect of your records. Without this option you would have to type in all the data again.

Now when you choose option 1 from the main menu—open new file—you're asked if you want to add fields to an old file. Answer Y and the program LOADs in the start of your old file and displays its structure showing the field names and lengths, the number of records used and the maximum allowed. You are told if you cannot add any more fields, otherwise you simply enter the number of extra fields and enter their names and lengths. You are then told how many records you can use with the new structure—take care if this is less than the number you have as the extra will be lost. The program then LOADs in the rest of the file. To fill in the new fields, View the records, then use Amend to fill in the gaps.

Delete Lines 1ØØØ to 111Ø then add:

```
8 *OPT2,1
15Ø PROCCREATE:GOTO 3Ø
2ØØ A$="N":PROCLOAD:GOTO 3Ø
1ØØØ DEF PROCCREATE
1Ø1Ø CLS
1Ø2Ø PRINTTAB(13,12)"ARE YOU SURE"
1Ø3Ø IF GET$<>"Y" THEN ENDPROC
1Ø4Ø N%=1:R%=Ø:A=Ø
1Ø5Ø CLS:PRINT"DO YOU WANT TO ADD
   FIELDS TO AN OLD FILE(Y/N)": INPUTA$
1Ø6Ø IF A$="Y" THEN PROCLOAD ELSE
   113Ø
1Ø7Ø S%=R%
1Ø8Ø CLS:PRINT'"THIS WAS YOUR OLD
   STRUCTURE"
1Ø9Ø PRINT'"FIELD NAME□□□□
   □□LENGTH":PRINT
11ØØ FOR T=1 TO A:PRINTN$(T),
   A(T):NEXT
111Ø PRINT'"MAX NO.=";M%;"□
   □□□□□NO. USED=";N%-1
112Ø IF A=8 THEN PRINT:PRINT"YOU
   CAN'T ADD ANY MORE FIELDS":GOTO
   126Ø
113Ø PRINT:PRINT"HOW MANY FIELDS DO
   YOU WANT";:IF A$="Y" THEN
   PRINT"□NOW";
114Ø PRINT"(";A+1;" TO 8)"
115Ø AA=GET-48:IF AA<A+1 OR AA>8
   THEN 115Ø
116Ø PRINT"ENTER NEW FIELD NAMES &
   LENGTHS"
117Ø PRINT
118Ø FOR N=A+1 TO AA
119Ø PRINT'"Name of field□";N;
   "□":PROCINPUT(1Ø):IF $B%<>""
   THEN N$(N)=$B% ELSE 119Ø
12ØØ PRINT'"What is the max length
   of□";N$(N);"□";
121Ø INPUT T:IF T<>Ø THEN A(N)=T
122Ø IF A(N)>27 OR A(N)<1 THEN
   PRINT"OUT OF RANGE":GOTO 121Ø
123Ø TRL(N)=R%:R%=A(N)+1+R%
124Ø NEXT N:IF R%<11 THEN R%=11
125Ø A=AA
126Ø M%=INT((&7CØØ-HIMEM)/R%)
   -1:PRINT''"You can use up to□";M%;
   "□records":D%=INKEY(3ØØ)
127Ø IF A$="Y" THEN PROCLOAD2PART
128Ø ENDPROC
8Ø23 IFA$="Y" THEN ENDPROC
82ØØ DEF PROCLOAD2PART
82Ø5 PRINT"LOADING UP THE REST OF
   YOUR FILE"
821Ø IF N%>M% THEN N%=M%+1
822Ø FOR T=1 TO N%-1
823Ø FOR Q=Ø TO R%-1
824Ø IF Q<S% THEN ?(B%+R%*
   T+Q)=BGET#X ELSE ?(B%+R%*
   T+Q)=13
825Ø NEXT
826Ø NEXT
827Ø GOTO 8Ø6Ø
```

ERROR TRAPPING

Make these changes to the error-trapping routine. You can press ESCAPE at any time to return to the main menu:

```
125 PRINT'"PRESS ESCAPE TO RETURN TO
   THE MAIN MENU";
13ØØØ VDU3:IF ERR=17 THEN 3Ø
13Ø5Ø REPORT:PRINT"□AT LINE□";ERL
```

CHANGES FOR DISK

The BBC automatically defaults to disk drive if a disk interface is fitted. The only changes you have to make to the program are to delete Lines 3 and 8.

If you have upgraded your computer from tape to disk and want to transfer your data then follow these steps. First type *TAPE then LOAD the datafile from tape. Delete Line 2, RUN the program, LOAD the data, press ESCAPE, type *DISK then type PROCSAVE. This will transfer the data to disk, but note that if you have filled up a large number of records, then some data may be lost due to the disk filing system taking up part of the computer's memory. Now press ESCAPE again, reinstate Line 2, delete Lines 3 and 8 as above and SAVE the datafile program on disk. It may sound complicated, but each step is really quite straightforward.

CHANGES FOR THE ELECTRON

You'll need to change the &7CØØ in Line 126Ø to &6ØØØ and change Line 1 to:

```
1 MODE 6:M%=Ø:N%=1:CF=1
```

There are four extra routines for the Dragon and three for the Tandy. These are the continuous print option, a new search, a new sort, and a routine to convert the datafile to work with the Dragon Data disk drive.

CONTINUOUS PRINT

This option allows you to print out all of your records in one go. Choose the usual Print option then choose either C for continuous print or S for single print. The printout starts from the record you are viewing and goes on to the end, so if you want a list of all your records be sure to start from the beginning.

```
1Ø5Ø PRINT@385,"NUMBER OF FIELDS
   (1-8) ?";
1Ø6Ø IN$=INKEY$:IFN$<"1" OR
   IN$>"8" THEN1Ø6Ø
1Ø7Ø A=VAL(IN$):DIM A(A),N$(A)
5Ø7Ø IF D>NR AND G=1 THENG=Ø:
   CH=-1:CP=Ø ELSE IF D>NR THEN
   CP=Ø:GOTO523Ø
51Ø5 IF CP=1 GOSUB1ØØ4Ø:GOTO516Ø
521Ø GOSUB1ØØØØ:GOTO516Ø
6Ø25 IF CP=1 GOSUB1ØØ4Ø:GOTO6Ø8Ø
613Ø GOSUB1ØØØØ:IF CP=1 THEN 6Ø8Ø
   ELSE 6Ø3Ø
614Ø IF D>NR THEND=1:CP=Ø
1ØØØØ PRINT@451,"□□□CHECK
   PRINTER□□□□cONT";STRING$
   (36,32);
1ØØ1Ø IF INKEY$<>"C" THEN 1ØØ1Ø
```

```
10020 PRINT@451,"cONTINUOUS OR
   sINGLE RECORD ?";
10030 IN$=INKEY$:IF IN$<>"C" AND
   IN$<>"S" THEN10030
10035 IF IN$="C" THEN CP=1
```

SORT ANY FIELD

In the old datafile the records are sorted automatically on field one as you enter them. The new menu option '8—Reorganize fields' now lets you choose which field is the main control field. When you choose this option the records are resorted and the new control field is displayed at the top of each record.

```
105 PRINT@388,"8: — REORGANIZE
   FIELDS"
120 IN$=INKEY$:IFIN$<"1"ORIN$
   >"8" THEN120
150 ON IN GOSUB1000,2000,6000,
   5000,7000,8000,9000,11000
11000 IF A=1 THEN PRINT"□CANNOT
   REORGANIZE 1 FIELD !":FORK=1
   TO5000:NEXT:RETURN
11010 PRINT"□FIELD NO.","NAME"
11020 FORN=1TOA:PRINTN,N$(N):
   NEXT
11030 PRINT:PRINT" ENTER NUMBER OF
   NEW CONTROL□□□□□FIELD (2
   TO";A;")";:INPUT NC
11040 NC=INT(NC):IF NC<2 OR NC>A
   THENCLS:GOTO11010
11050 CLS:PRINT"□CHANGING AND
   SORTING FIELDS"
11060 T$=N$(1):N$(1)=N$(NC):
   N$(NC)=T$:T=A(1):A(1)=A(NC):
   A(NC)=T
11070 FORN=1TONR:T$=A$(N,1):
   A$(N,1)=A$(N,NC):A$(N,NC)=
   T$:NEXT
11080 FORN=1TONR-1:K=N
11090 FORJ=N+1 TO NR
11100 IF A$(J,1)<A$(K,1) THEN K=J
11110 NEXT:IF N<>K THEN FORC=1
   TOA:T$=A$(K,C):A$(K,C)=A$
   (N,C):A$(N,C)=T$:NEXT
11120 NEXT:RETURN
```

MULTIPLE SEARCH

The new search option lets you search for entries in more than one field at once. When you choose the search option from the main menu you are asked what you are searching for in each field. You can search as many fields as you like. You are then asked if you want all these things to be found at the same time. If you answer N then the program will display records where at least one of the conditions applies. An example will make this clearer. Say you kept details of the stock in a clothes shop and were looking for shirts in field 2 and blue in field 4. If you want all these to apply at once, the program prints out all records with blue shirts. If you answer N to the question you'll be given a list of all shirts and all blue items.

```
5000 PRINT@B,B$;"search";B$;
   "option";B$
5010 BT=0:PRINT:FORN=1TOA:PRINT
   "SEARCH FIELD";N;",□";
   N$(N),"FOR WHAT ?";
5020 LINEINPUTS$(N):IF S$(N)<>
   "" THEN BT=BT+1
5025 NEXT:IF BT=0 THEN RETURN
5027 IF BT<2 THEN BT=0:GOTO5060
5030 PRINT:PRINT"DO YOU WANT ALL
   THESE TO BE□□□□□□FOUND
   TOGETHER (Y/N) ?";
5040 IN$=INKEY$:IF IN$<>"Y" AND
   IN$<>"N" THEN 5040
5050 CLS:BT=0:IF IN$="Y" THEN BT=1
5090 FORZ=1TOA:PS=INSTR(A$(D,
   Z),S$(Z)):IF PS>0 AND BT=0 AND
   S$(Z)<>"" THEN
   Z=A:NEXT:GOTO5100
5093 IF PS=0 AND BT=1 THEN
   Z=A:NEXT:D=D+CH:GOTO5070
5096 NEXT:IF BT=0 THEN D=D+CH:
   GOTO5070
5230 CLS2:PRINT@2," NO RECORD
   WITH□□□□□OF THESE";:IF BT=1
   THEN PRINT@18,"ALL□"; ELSE
   PRINT@18,"ANY□";
5235 FORZ=1TOA:IF S$(Z)="" THEN 5245
5240 PRINT@96+Z*32,N$(Z);:PRINT
   @107+Z*32,S$(Z);
5245 NEXT:CP=0
```

Q+A

Is there any way of recovering my data if the program crashes?

Yes, luckily there is. When a program crashes the data is not lost immediately, but it will be if you try to RUN the program again as this clears all the variables. So here's what to do:

Press [BREAK] then type GOTO 100. If [BREAK] doesn't work on the Commodore, follow it with [RESTORE] then enter poke bg,0:poke bd,0 followed by GOTO 100.

The error-trapping routine given at the bottom of page 951 makes sure the program stops if anything goes wrong. If this happens type GOTO 30 to restart it.

Press [BREAK] then type GOTO 40.

CHANGES FOR DISK

The next set of changes allow the program to work with a Dragon Data disk drive, and is for the Dragon only. Don't enter the lines if you intend to carry on with tape because they overwrite the tape LOAD and SAVE.

Delete Lines 8060 to 8070, 8150 to 8200 from the tape LOAD using DEL 8060–8070 and DEL 8150–8200, and Lines 7090 to 7140 from the tape SAVE by typing DEL 7090–7140 before entering the new lines, and see below for the method of transferring data from tape to disk.

```
80 PRINT@292,"5 :—□□SAVE FILE"
90 PRINT@324,"6 :—□□LOAD FILE"
1130 NEXT:R=INT(10000/(5+5*A))
   -1:PRINT"□MAX NUMBER OF
   RECORDS=";R
7000 CLS:PRINT"□ENSURE DRIVE IS ON
   AND A DISK□□□IS INSERTED, THEN
   PRESS enter"
7010 IF INKEY$<>CHR$(13) THEN 7010
7020 PRINT:PRINT"□INPUT FILENAME
   ?";:LINEINPUT FI$
7030 IF LEFT$(FI$,1)<"A" OR
   LEFT$(FI$,1)>"Z" THEN 7020
7040 CREATE FI$:CLS6:PRINT@232,
   "□SAVING□";FI$;
7050 FWRITE FI$;R,",",A,",",NR
7060 FORN=1TOA:FWRITE FI$;N$
   (N):FWRITE FI$;A(N):NEXT
7070 FORC=1TONR:FORN=1TOA:
   FWRITEFI$;A$(C,N):NEXTN,C
7080 CLOSE:RETURN
8030 PRINT@65,"SELECT DISK, THEN PRESS
   enter"
8040 IF INKEY$<>CHR$(13) THEN 8040
8050 IF R>0 THENRUN9210
8080 PRINT:PRINT"□INPUT FILENAME
   ?";:LINEINPUTFI$
8090 IF LEFT$(FI$,1)<"A" OR
   LEFT$(FI$,1)>"Z" THEN 8080
8100 FREAD FI$,FROM0;R,A,NR
8110 DIMA(A),N$(A),A$(R,A)
8120 FORN=1TOA:FREAD FI$;N$(N):
   FREAD FI$;A(N):NEXT
8130 FORC=1TONR:FORN=1TOA:
   FLREADFI$;A$(C,N):NEXTN,C
8140 CLOSE:RETURN
```

If you want to transfer existing data from tape to disk, first LOAD the program and make the disk SAVE changes. Then LOAD in the data from tape and SAVE to disk. Then make the disk LOAD changes and SAVE the new program on disk.

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 31 ...

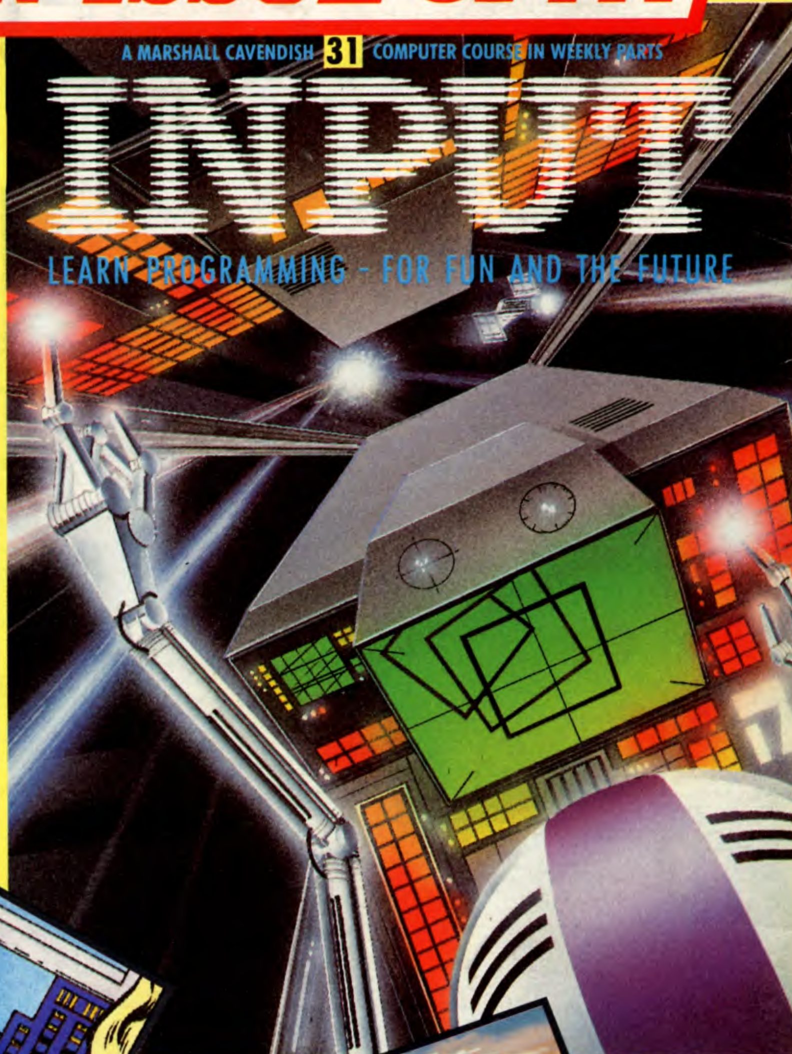
❑ *One of the first uses for computers was in code-breaking. Find out about the art of SENDING SECRET MESSAGES*

❑ *Enter part one of OTHELLO—a game in which you and the computer try to ensnare one another*

❑ *CLIFFHANGER continues with the routines before the game itself starts. This time, you add the THEME MUSIC*

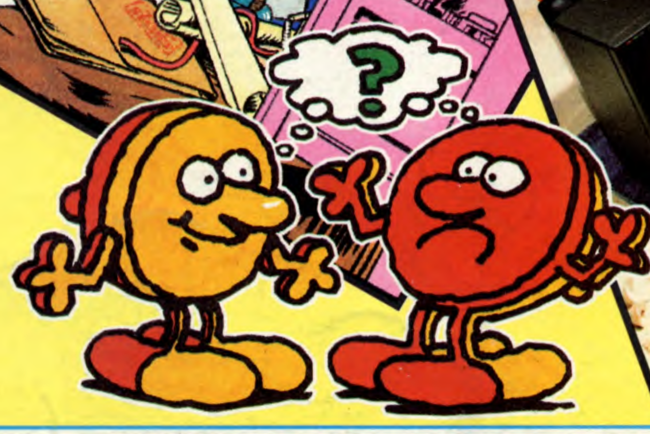
❑ *For high-level control, you often need to DETECT SEVERAL KEYPRESSES. Find out what's possible, and when*

❑ *On the ACORN machines, explore the sophisticated commands which control COLOUR MIXING AND FILLING*









ASK YOUR NEWSAGENT FOR INPUT